目錄

数据科学

本系列由《数据科学中的R语言》一书作者肖凯老师所著，现已开源。

目前共十篇：

- 数据挖掘导论和信贷模型
- 回归模型和房价预测
- 感知机和逻辑回归
- 决策树和集成学习
- 特征工程
- 参数调优
- 无监督学习
- 文本挖掘
- 神经网络
- 深度学习

欢迎fork and make contributions，仓库地址。

数据挖掘概览

一、引子：关于数据利用的故事

林彪发现敌军指挥部

在这个故事中，指挥官做了这样几件有关联的事情。一并称之为"数据闭环"。

- 不间断的收集战场数据；
- 基于某个指标抽象出战场实际；
- 对指标进行分析建模，发现一个机会；
- 基于分析结果进行战场决断，获取最大利益。

将指挥官换成CEO，就是商业数据闭环，基本上所有的商业数据应用，离不开这个套路。

你觉得指挥官建立了什么模型？先想想，别急着看答案

什么是数据闭环

- 度量商业行动：例如用一个指标测量用户响应率
- 识别商业机会：规划光棍节新产品或新活动，理解客户数据的波动，评价营销活动的结果
- 将数据转为知识：通过数据挖掘实施
- 基于知识行动：通常结合现有的业务流程，客户出现时推送信息，不同客户给予不同的资源

数据利用的四层境界

- 数据： 数据底层，原始数据的汪洋大海。形态：数据库。功能：直接取数
- 信息： 基于数据提炼得到的指标，新客有多少？老客有多少？老客都有什么特征，新客都有什么特征？形态：汇总报表。功能：指标提供，回答过去已经发生了什么的问题，业务人员运用得当也可以解决很多问题。
- 知识： 基于信息建立各指标之间的关系模型。什么情况下新客会转化为老客？模型结果。形态：模型，功能：回答为什么的问题，解释关系和因果，预测未来

- 智慧：将知识融入决策流程，将模型嵌入产品。我们要怎么做，才会让新客转化为老客。形态：数据产品。功能：控制未来

数据产品

- 数据产品就是给决策者提供行动信息的载体，例如
  - Amazon的商品推荐
  - 天气预报
  - Stock Market Predictions
  - Production Process Improvements
  - Health Diagnosis
  - Flu Trend Predictions
- 有些看起来也能提供决策者行动信息，如黄历，星相，但它们不是基于数据的洞察。

二、什么是数据挖掘

- 数据挖掘也称为知识发现。是一个去粗存精、去伪存真的过程。是从大量数据中提取、归纳有用知识的过程和方法。将其用于决策，可以提高人类的福利。
- 开普勒三大定律 开普勒的老师第谷收集了大量天文观测数据，但却是开普勒通过研究数据找到背后的规律
- 几个相关概念
  - 机器学习
  - 统计理论
  - 数据科学
  - 模式识别

	需要算法开发	不需要算法开发
需要数据开发	数据科学家	数据挖掘工程师
不需要数据开发	算法工程师	kaggle玩家

三、数据挖掘和我们的关系

为何需要数据挖掘：

- 如果没有数据，可以用什么决策？（直觉，经验归纳，逻辑推理，算命）
    - 需要数据，因为数据就是现实世界的历史痕迹，需要通过各种痕迹来推断未来，数据就是历史，挖掘就是归纳。
    - 数据太多，人脑无法直接处理。记录的数据越来越多，形式和来源都越来越复杂。自然产生的数据，人类社会产生的数据（社交网络，文本，图像，语音，视频......）
    - 所以需要挖掘工具和方法的帮助。

数据挖掘为什么火？

- 当前的各项前提条件已经具备。
    - 硬件价格的下降，使数据的存储和运算成本更低。
    - 个人和创业公司得以进入数据领域。
    - 开源软件工具和公开课分享使跨界更为容易。
    - 不同学科的壁垒被打破，可以较为容易的获得并学习其它学科的知识和工具，成为专业余人士。

数据挖掘和谁打交道

- 产品：侧重于底层数据框架搭建，数据报表开发，数据产品开发，例如淘宝的数据魔方的开发工作，这类工作需要很强的软件开发背景。
- 模型：侧重于对数据的研究，用统计理论或机器学习的方法对数据进行分析建模，例如广告的点击率分析建模，这类工作需要丰富的统计理论和模型算法知识。
- 美学：侧重于对数据的创作，用WEB技术进行数据可视化或者制作信息图，例如卫报的数据网站，需要很强的可视化能力和前端技术。
- 价值：侧重于数据中包含的商业价值研究，强调对专业领域的业务理解和交流沟通，例如咨询公司发布的商业分析报告，需要广泛的业务知识和商业敏感度。

应用领域有哪些：

- 商业零售
- 医疗
- 金融
- 太空探索

- 文字语音识别
- 下棋。。。

四、什么是数据挖掘模型？

- 一个关于模型的浅显例子：如何判断一个未切开的西瓜甜不甜？
- 可能的方案：
    - 准备N个西瓜
    - 设计M个变量或指标（特征工程），重量、花纹、茎叶的新鲜程度、卖家的位置...，切开前记录这些指标，记为X。
    - 切开后让n个人品尝打分(0表示不甜，1表示甜)，记为Y。
    - 使用其中一部分数据，结合分类算法对X和Y的关系进行建模。
    - 用剩下另一部分数据，检查模型的效果。
    - 把模型的逻辑写成一个APP放到应用市场上，持续收获数据，改进模型。
- 一个复杂的例子，如何判断一个老人未来是否会得痴呆症（思考思考）

商业中使用模型的例子：

- 银行的信用卡发放：

银行在信用卡发放的时候会进行审核。审核某个人的资格是否符合条件以授信。在传统的审核工作，这种事是人工来做的，申请人填一张表，写上个人的年龄、职业、收入等信息（X变量）。交给有经验的银行风控师，他们来进行评价，是发信用卡，还是不发信用卡。

但在互联网时代，这种人工审核就太慢了。互联网金融的崛起就是最明显的趋势。它将全网中关于个人的行为数据进行收集整合，其中有必然有一部分人已经在金融机构有过借贷行为，考察这种行为是否有违约，将其作为Y。将其它的行为数据作为X。这样就构成了一个可以喂到分类算法中的数据集。然后这个模型就可以用在未来的申请人身上，形成审核自动化系统。

- gmail垃圾邮件自动分类：

如果你点开自己的gmail邮箱，仔细观察会发现一个垃圾邮件的标签，它平时默默的为你挡下大量的垃圾信息，而又不去干扰你，实在是数据产品的典范。那么如果你来做这种垃圾邮件的自动分类，要怎么做的呢？

如果我们考虑简单些，抛开一封电邮中的其它信息（发件人，IP...），而只取文本信息的话，这个问题就转为一个文本分类的问题。

文本的分析难点在于：文本不是给计算机阅读的，它有复杂的语言结构（语法、语义、语用），但语言中依然存在统计规律（统计语言模型）。

一个简单的文本分类模型：判断一封邮件是否垃圾邮件

```
- 收集N个邮件
- 从邮件中提取指标（分词，空间向量模型），构成文档-词项矩阵
- 人工标注这些邮件是否垃圾邮件
- 用一部分数据，结合算法对X和Y的关系进行建模。
- 用剩下的数据，检查模型的效果。
```

五、数据挖掘日常工作有哪些

- 项目讨论和规划，就是开会。这方面工作目的主要是明确业务问题。是不是可以做？大概可以怎么做？确定了业务问题之后，需要将这个业务问题翻译成一个数据问题。

- 项目准备，准备开工干活了。这方面是最为繁琐也最容易出错的地方。需要和数据仓库的同学配合取得必要的数据，探索理解数据的业务意义，评估数据质量，根据项目需要对数据进行整理转换，做大量的特征工程的工作。

- 项目实施，即数据建模，开始拷打数据了。选择尝试不同的模型算法，从数据中得到需要的结果，然后从不同方面评价效果怎么样。

- 项目结束，交付结果。确定模型如何部署，并实施部署工作。这种部署就是模型的应用，多数情况下是将结果回写到数据库中。同时结果交付给需求方，写最终的项目报告，归档所有文件。

- 阅读文献，方法研究。在比较空闲的时间，或者遇到难题的时候，都需要去找巨人的肩膀依靠一下。

一个典型的步骤流程

- 商业理解：理解业务目标和需求，并转化为数据挖掘可理解的问题定义。建模师会参加业务组的会议，主要是了解收集业务需求。
- 数据理解：筛选目标数据，检验数据质量，探索数据特征，评估可用数据。建

模师会将一些初步结果呈现给业务组，得到进一步反馈。
- 数据准备：通过清洗，集成，变换，归约等处理方法构造最终数据集合。建模师开始疯狂的写SQL类的脚本去洗数据。
- 模型建立：选择和应用各种机器学习或统计方法、构建模型并调校各种参数。建模师进入炼丹阶段，期待能有好的结果。
- 模型评价：结合最初的商业目标评价并解释模型，评估其可能的商业效果。建模师将模型结果和业务团队进行沟通。
- 模型部署：按用户习惯方式实施并发布模型，提供分析结论，并持续跟踪。建模师将模型上线，监测性能。

六、数据挖掘的任务模式

- 分类 Classification [Predictive]
- 聚类 Clustering [Descriptive]
- 关联规则 Association Rule Discovery [Descriptive]
- 序列挖掘 Sequential Pattern Discovery [Descriptive]
- 回归 Regression [Predictive]
- 异常检测 Deviation Detection [Predictive]

分类方法的应用

精准化营销

- 问题: 准备发售iphone新品了，哪些用户可能会买？
- 方法:
  - 找到相似产品的用户行为数据，观察是否购买作为目标变量
  - 收集这些用户的行为特征，作为模型的解释变量

聚类方法的应用

用户分群:

- 问题: 如何将用户分成若干组，然后针对不同用户组进行营销活动
- 方法:
  - 收集用户的基本社会特征和行为特征

  - 根据用户的相似程度进行聚类分组
  - 根据同组的用户购买行为判断分群的效果

关联规则应用

相似商品推荐

- 关联规则是一种规律 {面包干, … } --> {薯片}
- 买了面包干的用户往往也会去买薯片，可以利用这种关联规则，将若干商品一起放在货架上捆绑销售
- 不同的关联商品，暗示了不同的消费场景

异常检测应用

- 从正常行为中发现不正常的行为模式
- 信用卡欺诈
- 网络入侵

七、需要掌握哪些技能

- 有形的技能:
  - 理论：气宗。 例如统计理论、机器学习算法。个人体会精通理论后再做数据工作就如汤泼雪。我也承认学习理论是艰难的，但是一定要在年轻的时候读最难的书。《数学之美》中谈到，技术分为术和道两种，具体的做事方法是术，做事的原理和原则是道，只追求术的人工作很辛苦，只有掌握了道才能永远游刃有余。
  - 工具：剑宗。理论不用在产品上就是王语焉的学院派。从理论到产品，需要掌握各种工具。这类工具用得熟了能事半功倍，例如R、python、SQL、hadoop这类。学习工具和学习语言一样，都要多读多写，模仿揣摩，就可以运用自如。不过迷信工具是没有意义的，没有最好的工具，只有最合适的工具。如果你是独孤求败，可以玩玄铁剑，如果你是东方不败，可以玩绣花针。
  - 经验：实战。有内力有剑法，就需要下山了。对战最强悍的对手，才能让你的内力剑法融为一体。做项目，在工作解决难题，才是长进最快的。
- 无形的气质:

- 好奇，好奇心和兴趣是从数据中得到洞察的驱动力。有好奇心的人才会对数据有持续的热情。
- 创造，兵无常势，数据的工作都是千差万别的，虽然可以依靠一些老的经验做些照猫画虎的事。但最好还是需要根据不同的项目情况来做出判断。独立思考和创造让你走得更远。
- 求败，创造、前沿、探索性的工作，一定会有失败，快速失败，快速学习，不断修正，能够败中求胜。

如何培养数据挖掘的技能

- 两个字：自学。“知识与耐心，是击败强者的唯一方法。”
  - 通过阅读来学习。包括了阅读经典的理论教材、代码、论文、上公开课。
  - 通过牛人来学习。包括同行的聚会、讨论、大牛的博客、微博、twitter、RSS。
  - 通过练习来学习。包括代码练习题、参加kaggle比赛、解决实际工作中的难题。
  - 通过分享来学习。包括自己写笔记、写博客、和同事分享交流、培训新人。

八、经验之谈

建模中的坑

- 建模过程的问题
  - 缺乏业务问题的沟通和理解
  - 只关注训练数据或只过于相信数据
  - 只依赖于一种技术
  - 错误的变量输入
- 挖掘结果不真实
  - 模型结果不代表任何规律
  - 模型训练集可能不反映真正的总体
  - 数据的详细程度有误
- 挖掘结果没有用
  - 挖掘结果众所周知

 - 挖掘结果不可用于决策

一点心得

- 提问题比回答问题更重要 一个具体的业务痛点是数据挖掘的起点，精心计划流程步骤，业务知识贯穿挖掘建模的每个阶段。
- 对数据持谨慎的态度 数据很可能出错，数据整理占据大部分的工作时间。
- 数据本身仅能用于描述历史 不能展现因果，也不能预知未来。
- 数据价值体现在落地应用 数据挖掘价值并不取决于模型的准确或稳定，取决于背后的决策组织。
- 不同的指标和模型都有其适用范围 随时间环境变化，所有的模式都会改变，不断尝试，不断修正。
- 工作中的文档化和自动化

一个综合案例的建模步骤

- 问题定义
- 数据探索
- 特征工程
- 建模和评估

问题定义

我们使用kaggle上一个经典的问题做为案例示范，即判断一个贷款者在后续两年内是否会违约的概率。

需要思考的关键问题

- 损失是如何发生的?
- 违约的人有哪些特点?
- 违约的占比有多少？
- 如何能改善我们的损失？

数据下载

http://www.kaggle.com/c/GiveMeSomeCredit

变量的意义

- SeriousDlqin2yrs
  - 用户在后续两年内出现90天以上的还款逾期，这是目标变量，以下都是解释变量
- RevolvingUtilizationOfUnsecuredLines
  - 信用卡借款占比
- age
  - 借款人年龄
- NumberOfTime30-59DaysPastDueNotWorse
  - 过去有逾期30-59天还款的次数
- DebtRatio
  - 月度生活成本占月收入的比率
- MonthlyIncome
  - 月收入
- NumberOfOpenCreditLinesAndLoans
  - 包括车贷房贷在内的贷款笔数
- NumberOfTimes90DaysLate
  - 过去有逾期90天以上还款的次数
- NumberRealEstateLoansOrLines
  - 房贷的信贷次数
- NumberOfTime60-89DaysPastDueNotWorse
  - 过去有逾期60-89天还款的次数
- NumberOfDependents
  - 家庭中需要抚养者的人数

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
df = pd.read_csv("data/credit-training.csv") #读取数据
df.head()
```

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	5
0	1	0.766127	45	2
1	0	0.957151	40	0
2	0	0.658180	38	1
3	0	0.233810	30	0
4	0	0.907239	49	1

```
df.shape # 一共15万条记录
```

```
(150000, 11)
```

数据探索

```
df.info() # 有缺失值
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 11 columns):
SeriousDlqin2yrs                        150000 non-null int64
RevolvingUtilizationOfUnsecuredLines    150000 non-null float64
age                                     150000 non-null int64
NumberOfTime30-59DaysPastDueNotWorse    150000 non-null int64
DebtRatio                               150000 non-null float64
MonthlyIncome                           120269 non-null float64
NumberOfOpenCreditLinesAndLoans         150000 non-null int64
NumberOfTimes90DaysLate                 150000 non-null int64
NumberRealEstateLoansOrLines            150000 non-null int64
NumberOfTime60-89DaysPastDueNotWorse    150000 non-null int64
NumberOfDependents                      146076 non-null float64
dtypes: float64(4), int64(7)
memory usage: 12.6 MB
```

```
df.describe() # 描述性统计
```

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	
count	150000.000000	150000.000000	150(
mean	0.066840	6.048438	52.2
std	0.249746	249.755371	14.7
min	0.000000	0.000000	0.00
25%	0.000000	0.029867	41.(
50%	0.000000	0.154181	52.(
75%	0.000000	0.559046	63.(
max	1.000000	50708.000000	109

```
df.SeriousDlqin2yrs.value_counts()
```

```
0    139974
1     10026
Name: SeriousDlqin2yrs, dtype: int64
```

```
df.SeriousDlqin2yrs.mean() # 平均严重拖欠
```

```
0.06684
```

```
df.NumberOfDependents.unique() # 受抚养者, 查看有多少取值
```

```
array([  2.,   1.,   0.,  nan,   3.,   4.,   5.,   6.,   8.,   7
.,  20.,
        10.,   9.,  13.])
```

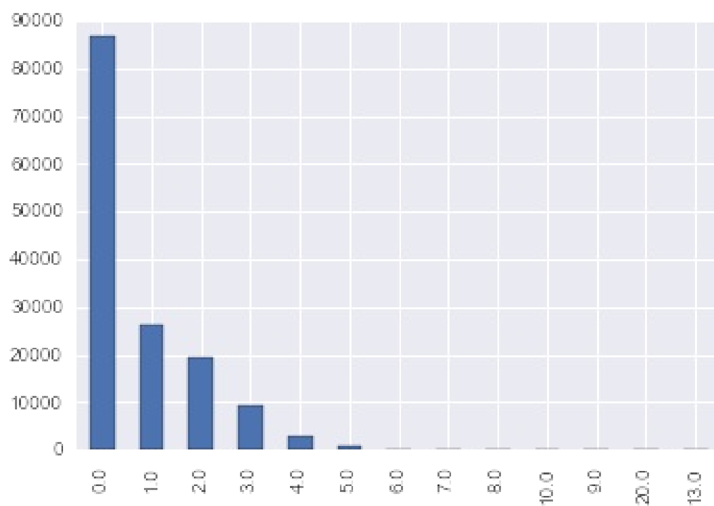
```
df.NumberOfDependents.value_counts()
# 观察频数表，一半都没有, 有的话集中1-3个, 再往上较少
```

```
0.0     86902
1.0     26316
2.0     19522
3.0      9483
4.0      2862
5.0       746
6.0       158
7.0        51
8.0        24
10.0        5
9.0         5
20.0        1
13.0        1
Name: NumberOfDependents, dtype: int64
```

```
df.groupby("SeriousDlqin2yrs").mean()
```

	RevolvingUtilizationOfUnsecuredLines	age
SeriousDlqin2yrs		
0	6.168855	52.751375
1	4.367282	45.926591

```
pd.value_counts(df.NumberOfDependents).plot(kind='bar');
```




```
pd.crosstab(df.NumberOfTimes90DaysLate, df.SeriousDlqin2yrs)
# 计算交叉频数表
```

SeriousDlqin2yrs	0	1
NumberOfTimes90DaysLate		
0	135108	6554
1	3478	1765
2	779	776
3	282	385
4	96	195
5	48	83
6	32	48
7	7	31
8	6	15
9	5	14
10	3	5
11	2	3
12	1	1
13	2	2
14	1	1
15	2	0
17	0	1
96	1	4
98	121	143

```
pd.crosstab(df.age, df.NumberOfDependents)
# 年龄与受抚养者的关系
# 岁数越大抚养者会多
```

NumberOfDependents	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0
age								
0	0	0	1	0	0	0	0	0
21	148	3	1	0	0	0	0	0

22	385	7	2	2	0	0	0	0
23	550	33	13	3	0	0	0	0
24	689	48	19	3	1	0	0	0
25	774	91	31	7	5	1	0	0
26	946	128	56	14	4	0	0	0
27	1001	192	53	32	4	0	0	0
28	1142	210	114	45	8	1	0	0
29	1195	254	145	53	14	1	1	0
30	1337	288	178	72	14	7	1	1
31	1314	344	245	91	18	4	0	0
32	1207	380	275	106	53	3	0	0
33	1254	449	315	140	50	10	2	0
34	1152	389	360	145	65	11	1	0
35	1136	418	398	184	58	15	3	1
36	1139	445	486	202	50	15	5	1
37	1088	488	541	274	77	18	2	0
38	1113	484	604	283	84	26	3	0
39	1202	544	691	383	111	27	4	1
40	1185	570	758	391	104	43	5	1
41	1193	517	791	445	113	30	4	0
42	1159	526	755	427	142	38	7	2
43	1205	556	789	431	156	33	9	4
44	1209	574	854	461	133	32	7	2
45	1363	608	828	457	165	35	11	3
46	1373	695	876	498	165	35	8	8
47	1439	672	863	491	148	46	9	0
48	1509	771	791	459	157	37	11	3
49	1632	717	790	439	143	46	8	6
...	...	...	...	...	...	...	...	...
74	1164	184	23	4	2	0	0	1

75	981	167	17	3	0	0	0	0
76	933	165	16	0	0	0	0	0
77	890	129	11	2	0	0	0	0
78	837	137	13	2	1	0	0	0
79	770	131	10	4	0	0	0	0
80	704	109	7	2	0	0	0	0
81	629	76	6	1	0	0	0	0
82	527	69	1	0	1	0	0	0
83	417	44	2	0	0	0	0	0
84	406	32	1	0	0	0	0	0
85	403	30	2	0	0	0	0	0
86	318	46	2	0	1	0	0	0
87	279	29	1	1	0	0	0	0
88	244	20	1	0	0	0	0	0
89	230	11	1	0	0	0	0	0
90	148	19	0	0	0	0	0	0
91	119	10	0	0	0	0	0	0
92	75	7	0	0	0	0	0	0
93	67	3	0	0	0	0	0	0
94	33	2	0	0	0	0	0	0
95	33	4	0	0	0	0	0	0
96	12	2	0	0	0	0	0	0
97	11	1	0	0	0	0	0	0
98	5	0	0	0	0	0	0	0
99	5	0	0	0	0	0	0	0
101	3	0	0	0	0	0	0	0
102	2	1	0	0	0	0	0	0
103	3	0	0	0	0	0	0	0
107	1	0	0	0	0	0	0	0

84 rows × 13 columns

清洗数据

```
import re
# 将名字都改为 snake_case
def camel_to_snake(column_name):
    """
    converts a string that is camelCase into snake_case
    """
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', column_name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
camel_to_snake("javaLovesCamelCase")
```

```
'java_loves_camel_case'
```

```
df.columns = [camel_to_snake(col) for col in df.columns]
df.columns.tolist()
```

```
['serious_dlqin2yrs',
 'revolving_utilization_of_unsecured_lines',
 'age',
 'number_of_time30-59_days_past_due_not_worse',
 'debt_ratio',
 'monthly_income',
 'number_of_open_credit_lines_and_loans',
 'number_of_times90_days_late',
 'number_real_estate_loans_or_lines',
 'number_of_time60-89_days_past_due_not_worse',
 'number_of_dependents']
```

```
from sklearn.neighbors import KNeighborsRegressor
income_imputer = KNeighborsRegressor(n_neighbors=1)

# 数据分为两部分，有缺失的和无缺失的，用无缺失的数据建立模型来判断缺失数据的
可能取值
train_w_monthly_income = df[df.monthly_income.isnull()==False]
train_w_null_monthly_income = df[df.monthly_income.isnull()==True
]
```

```
cols = ['number_real_estate_loans_or_lines', 'number_of_open_cre
dit_lines_and_loans']
income_imputer.fit(train_w_monthly_income[cols], train_w_monthly
_income.monthly_income)
# 用房产贷款次数以及未结束贷款次数来训练月收入
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='mink
owski',
          metric_params=None, n_jobs=1, n_neighbors=1, p=2,
          weights='uniform')
```

```
new_values = income_imputer.predict(train_w_null_monthly_income[
cols])
# 再用模型预测缺失值中的月收入
```

```
train_w_null_monthly_income.ix[:,'monthly_income']=new_values
# imputation
```

```
df_imputed = train_w_monthly_income.append(train_w_null_monthly_
income)
df_imputed.shape # 填充缺失值 done
```

```
(150000, 11)
```

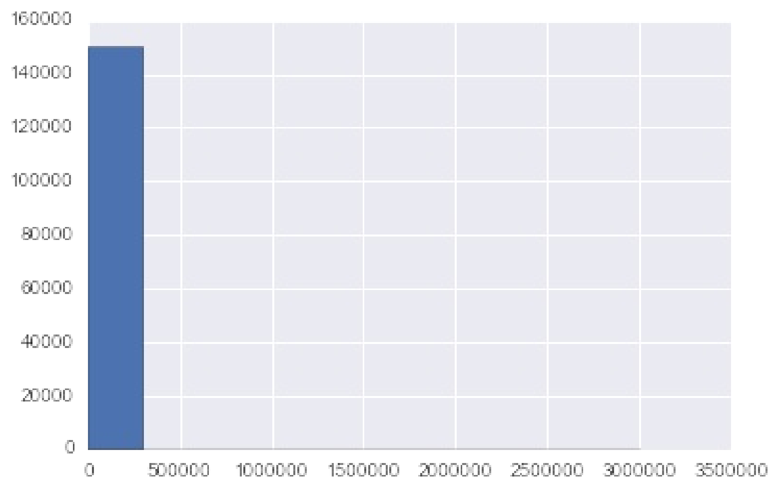
```
df_imputed.ix[df_imputed.number_of_dependents.isnull(),'number_o
f_dependents'] = -1
```

```
df_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150000 entries, 0 to 149997
Data columns (total 11 columns):
serious_dlqin2yrs                                 150000 non-null i
nt64
revolving_utilization_of_unsecured_lines          150000 non-null f
loat64
age                                               150000 non-null i
nt64
number_of_time30-59_days_past_due_not_worse       150000 non-null i
nt64
debt_ratio                                        150000 non-null f
loat64
monthly_income                                    150000 non-null f
loat64
number_of_open_credit_lines_and_loans             150000 non-null i
nt64
number_of_times90_days_late                       150000 non-null i
nt64
number_real_estate_loans_or_lines                 150000 non-null i
nt64
number_of_time60-89_days_past_due_not_worse       150000 non-null i
nt64
number_of_dependents                              150000 non-null f
loat64
dtypes: float64(4), int64(7)
memory usage: 13.7 MB
```

特征工程

```
df_imputed.monthly_income.hist();
```




```
def cap_values(x, cap):
    if x > cap:
        return cap
    else:
        return x
# 设定上限
df_imputed.monthly_income = df_imputed.monthly_income.apply(lamb
da x: cap_values(x, 15000))
```

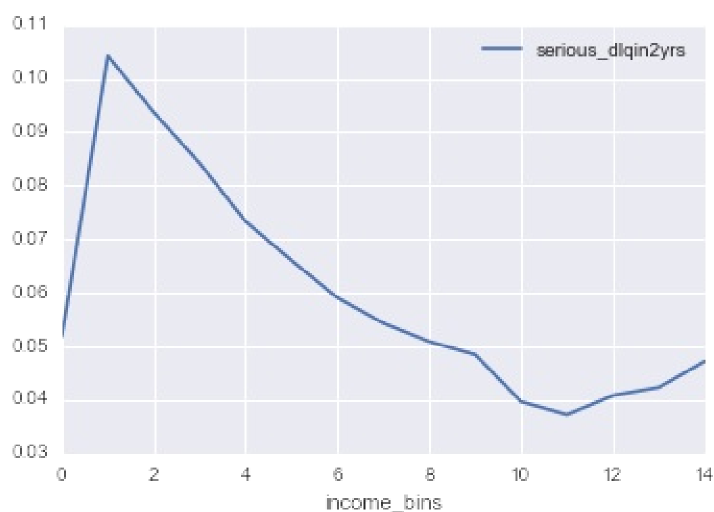
```
# 变量离散化，分为15个bin
df_imputed['income_bins'] = pd.cut(df_imputed.monthly_income, bi
ns=15, labels=False)
pd.value_counts(df_imputed.income_bins)
```

```
3     23168
4     19944
5     15583
6     14475
2     14038
7     10766
8      8609
14     7775
9      7672
1      7504
10     6298
0      4994
11     4454
12     2547
13     2173
Name: income_bins, dtype: int64
```

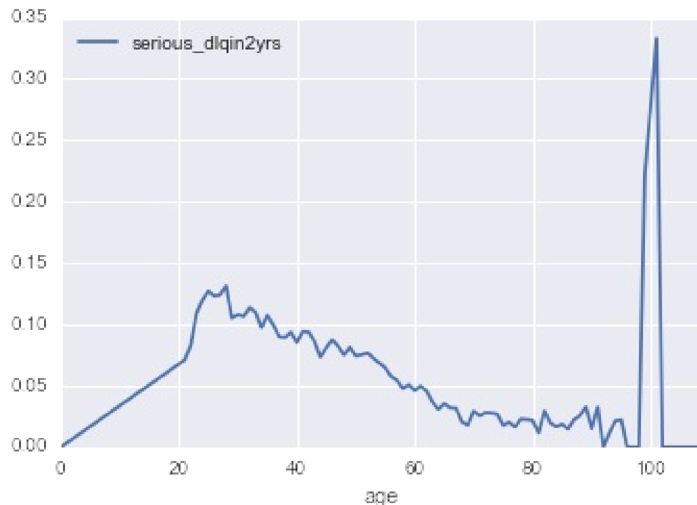
```
df_imputed[["income_bins", "serious_dlqin2yrs"]].groupby("income_bins").mean()
# 每个月收入分类中统计default平均频数
```

	serious_dlqin2yrs
income_bins	
0	0.051862
1	0.104211
2	0.093674
3	0.084168
4	0.073305
5	0.066033
6	0.059067
7	0.054338
8	0.050877
9	0.048488
10	0.039695
11	0.037270
12	0.040832
13	0.042338
14	0.047203

```
# 画出图来, 可以明显发现在1-2个 bin中的 default 最多
cols = ["income_bins", "serious_dlqin2yrs"]
df_imputed[cols].groupby("income_bins").mean().plot();
```



```
# 以年龄来看是20-30岁最高，另外就是100岁前后，可能是去世了?
cols = ['age', 'serious_dlqin2yrs']
age_means = df_imputed[cols].groupby("age").mean()
age_means.plot();
```




```
mybins = [0] + range(20, 80, 5) + [120]
df_imputed['age_bin'] = pd.cut(df_imputed.age, bins=mybins)
pd.value_counts(df_imputed['age_bin'])
```

```
(45, 50]     18829
(50, 55]     17861
(55, 60]     16945
(60, 65]     16461
(40, 45]     16208
(35, 40]     13611
(65, 70]     10963
(30, 35]     10728
(75, 120]    10129
(25, 30]      7730
(70, 75]      7507
(20, 25]      3027
(0, 20]          0
dtype: int64
```

```
from sklearn.preprocessing import StandardScaler
# Standardize features by removing the mean and scaling to unit variance

# 将月收入标准化
df_imputed['monthly_income_scaled'] = StandardScaler().fit_transform(df_imputed.monthly_income.reshape(-1,1))
```

建模和评估

```
df_imputed.columns
```

```
Index([u'serious_dlqin2yrs', u'revolving_utilization_of_unsecured_lines',
       u'age', u'number_of_time30-59_days_past_due_not_worse', u'debt_ratio',
       u'monthly_income', u'number_of_open_credit_lines_and_loans',
       u'number_of_times90_days_late', u'number_real_estate_loans_or_lines',
       u'number_of_time60-89_days_past_due_not_worse', u'number_of_dependents',
       u'income_bins', u'age_bin', u'monthly_income_scaled'],
      dtype='object')
```

```
# 特征
features = ['revolving_utilization_of_unsecured_lines',
 'age',
 'number_of_time30-59_days_past_due_not_worse',
 'debt_ratio',
 'monthly_income',
 'number_of_open_credit_lines_and_loans',
 'number_of_times90_days_late',
 'number_real_estate_loans_or_lines',
 'number_of_time60-89_days_past_due_not_worse',
 'number_of_dependents',
 'income_bins',
 'age_bin',
 'monthly_income_scaled']
```

```
X = pd.get_dummies(df_imputed[features], columns = ['income_bins'
, 'age_bin'])
# dummy var
```

```
print X.columns.tolist()
print X.shape
```

```
['revolving_utilization_of_unsecured_lines', 'age', 'number_of_t
ime30-59_days_past_due_not_worse', 'debt_ratio', 'monthly_income
', 'number_of_open_credit_lines_and_loans', 'number_of_times90_d
ays_late', 'number_real_estate_loans_or_lines', 'number_of_time6
0-89_days_past_due_not_worse', 'number_of_dependents', 'monthly_
income_scaled', 'income_bins_0', 'income_bins_1', 'income_bins_2
', 'income_bins_3', 'income_bins_4', 'income_bins_5', 'income_bi
ns_6', 'income_bins_7', 'income_bins_8', 'income_bins_9', 'incom
e_bins_10', 'income_bins_11', 'income_bins_12', 'income_bins_13'
, 'income_bins_14', 'age_bin_(0, 20]', 'age_bin_(20, 25]', 'age_
bin_(25, 30]', 'age_bin_(30, 35]', 'age_bin_(35, 40]', 'age_bin_
(40, 45]', 'age_bin_(45, 50]', 'age_bin_(50, 55]', 'age_bin_(55,
 60]', 'age_bin_(60, 65]', 'age_bin_(65, 70]', 'age_bin_(70, 75]
', 'age_bin_(75, 120]']
(150000, 39)
```

```
y = df_imputed.serious_dlqin2yrs # target
```

```
from sklearn.cross_validation import train_test_split
train_X, test_X, train_y, test_y = train_test_split(X, y ,train_
size=0.7,random_state=1)
# 70% 是 train data
```

```
print train_X.shape
print test_X.shape
```

```
(105000, 39)
(45000, 39)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoo
stingClassifier
```

```
clf = LogisticRegression() # logit 逻辑回归
clf.fit(train_X,train_y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_int
ercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', t
ol=0.0001,
          verbose=0, warm_start=False)
```

```
clf.predict_proba(test_X) # 预测概率
```

```
array([[ 0.92834155,  0.07165845],
       [ 0.96492419,  0.03507581],
       [ 0.95753197,  0.04246803],
       ...,
       [ 0.96021551,  0.03978449],
       [ 0.87166534,  0.12833466],
       [ 0.9777555 ,  0.0222445 ]])
```

```
from sklearn.metrics import roc_curve, roc_auc_score, classifica
tion_report, confusion_matrix
# 评价方法
```

```
preds = clf.predict(test_X)
confusion_matrix(test_y, preds)
```

```
array([[41880,    88],
       [ 2908,   124]])
```

```
print classification_report(test_y, preds, labels=[0, 1])
```

```
             precision    recall  f1-score   support

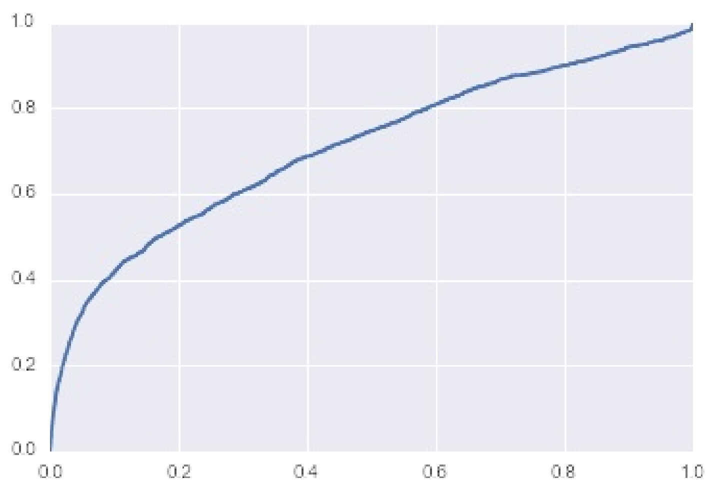
          0       0.94      1.00      0.97     41968
          1       0.58      0.04      0.08      3032

avg / total       0.91      0.93      0.91     45000
```

```
pre = clf.predict_proba(test_X)
roc_auc_score(test_y,pre[:,1])
```

```
0.7078206950866951
```

```
fpr, tpr, thresholds = roc_curve(test_y,pre[:,1])
plt.plot(fpr,tpr,);
```




```
# 随机森林分类器
clf = RandomForestClassifier()
clf.fit(train_X,train_y)
preds = clf.predict(test_X)
print classification_report(test_y, preds, labels=[0, 1])
```

```
             precision    recall  f1-score   support

          0       0.94      0.99      0.97     41968
          1       0.51      0.15      0.24      3032

avg / total       0.91      0.93      0.92     45000
```

```
pre = clf.predict_proba(test_X)
roc_auc_score(test_y,pre[:,1])
```

```
0.7867644886115015
```

练习：你来改进这个挖掘的结果，使auc可以比0.78更大

```
df_imputed.to_csv('df_imputed',index = False)
```

Sections

Exploring and visualizing the Housing dataset

[back to top]

波士顿房价数据

Source: https://archive.ics.uci.edu/ml/datasets/Housing

Attributes:

```
1. CRIM      per capita crime rate by town 每个城镇人均犯罪率
2. ZN        proportion of residential land zoned for lots over
                 25,000 sq.ft. 超过25000平方尺用地划为居住用地的百分比
3. INDUS     proportion of non-retail business acres per town 非
零售商用地百分比
4. CHAS      Charles River dummy variable (= 1 if tract bounds
                 river; 0 otherwise) 是否被河道包围
5. NOX       nitric oxides concentration (parts per 10 million)
氮氧化物浓度
6. RM        average number of rooms per dwelling 住宅平均房间数目
7. AGE       proportion of owner-occupied units built prior to 1
940 1940年前建成自用单位比例
8. DIS       weighted distances to five Boston employment centre
s 5个波士顿就业服务中心的加权距离
9. RAD       index of accessibility to radial highways 无障碍径向
高速公路指数
10. TAX      full-value property-tax rate per $10,000 每万元物业税
率
11. PTRATIO  pupil-teacher ratio by town 小学师生比例
12. B        1000(Bk - 0.63)^2 where Bk is the proportion of bla
cks
                 by town 黑人比例指数
13. LSTAT    % lower status of the population 低层人口比例
14. MEDV     Median value of owner-occupied homes in $1000's 业主
自住房屋中值 （要预测的变量）
```

```
# 读取数据
import pandas as pd
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learnin
g-databases/housing/housing.data',
                 header=None, sep='\s+')  # 分隔符为空格

df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS',
              'NOX', 'RM', 'AGE', 'DIS', 'RAD',
              'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622

数据分析的第一步是进行探索性数据分析 **(Exploratory Data Analysis, EDA)**，理解变量的分布与变量之间的关系。

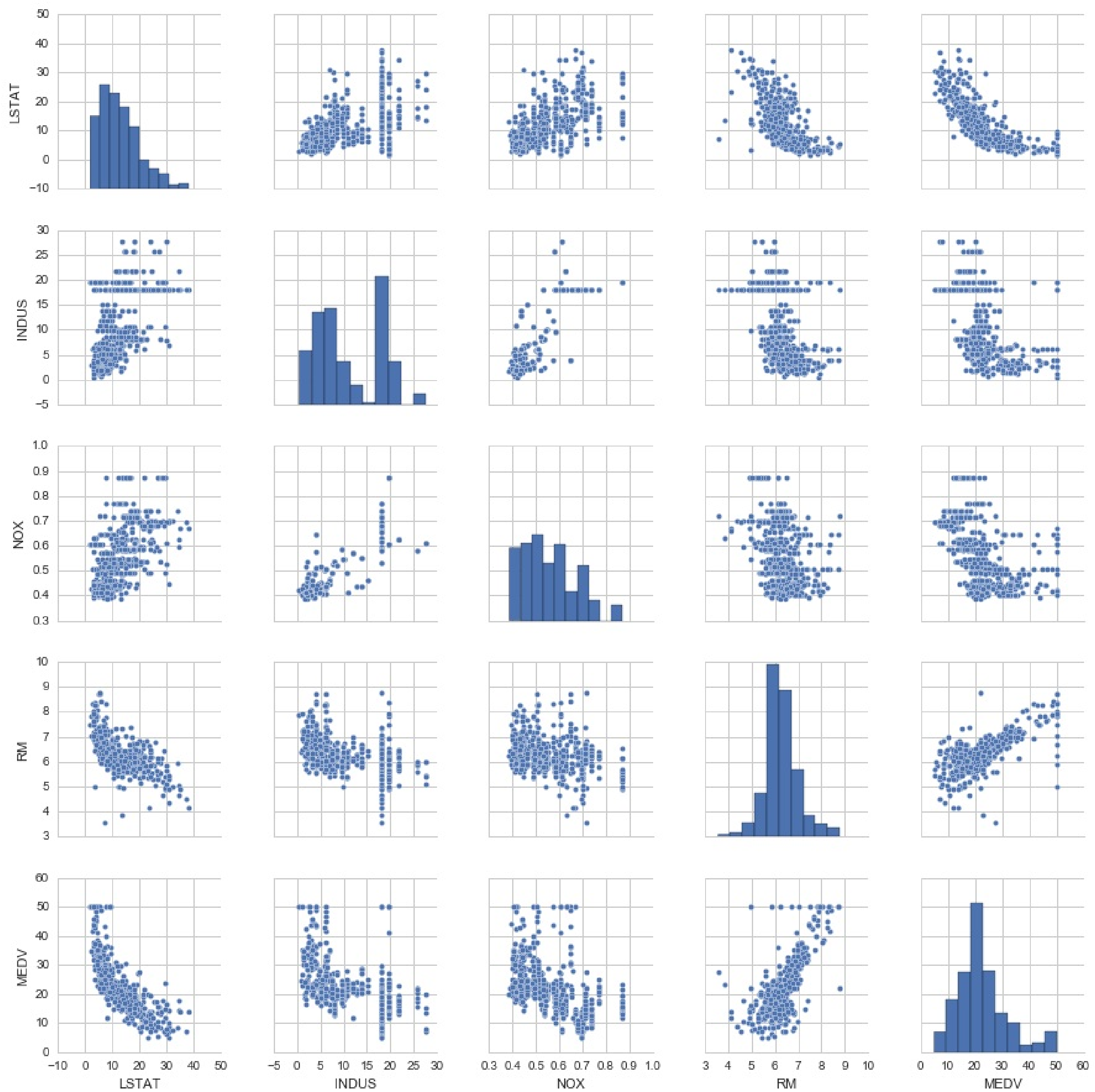
```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid', context='notebook')  # 设定样式，还原可用 sns.reset_orig

# MEDV 是目标变量，为了方便演示，只挑 4 个预测变量
cols = ['LSTAT', 'INDUS', 'NOX', 'RM', 'MEDV']

# scatterplot matrix, 对角线上是变量分布的直方图，非对角线上是两个变量的散点图
sns.pairplot(df[cols], size=2.5)
plt.tight_layout()

# 用下面这行代码可以存储图片到硬盘中
# plt.savefig('./figures/scatter.png', dpi=300)
```

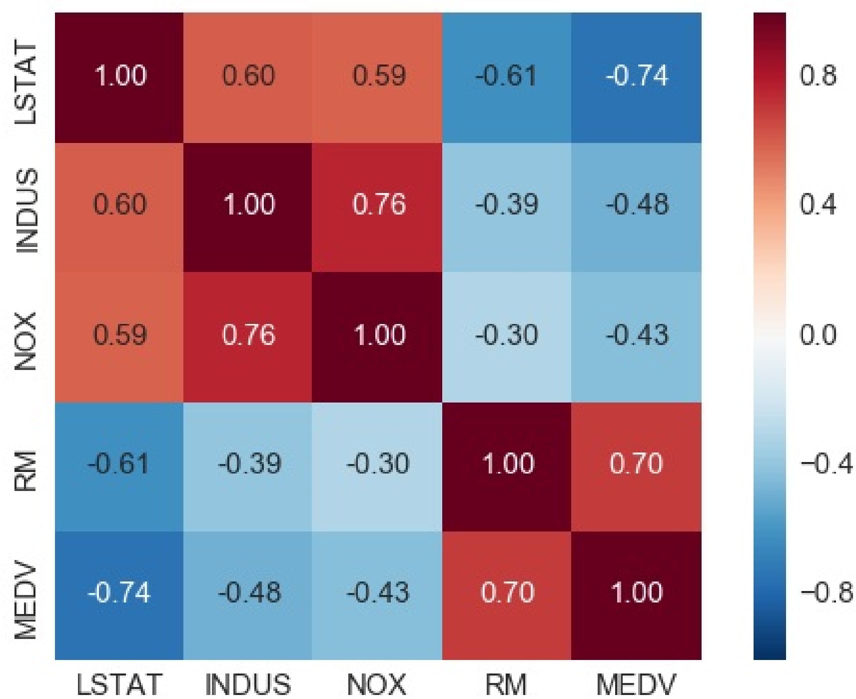


从图中看出

- RM 和 MEDV 似乎是有线性关系的
- MEDV 类似 normal distribution

```
# correlation map
import numpy as np
cm = np.corrcoef(df[cols].values.T)  # 计算相关系数
sns.set(font_scale=1.5)

# 画相关系数矩阵的热点图
hm = sns.heatmap(cm,
        annot=True,
        square=True,
        fmt='.2f',
        annot_kws={'size': 15},
        yticklabels=cols,
        xticklabels=cols)
plt.tight_layout()
# plt.savefig('./figures/corr_mat.png', dpi=300)
```



- 对与 MEDV correlation 高的变量感兴趣, LSTAT 最高(-0.74), 其次是 RM (0.7)
- 但从之前的图看出 MEDV 与 LSTAT 呈非线性关系，而与 RM 更呈线性关系，所以下面选用 RM 来演示简单线性回归

```
sns.reset_orig()
```

Implementing a simple regression model - Ordinary least squares

Solving regression parameters with gradient descent

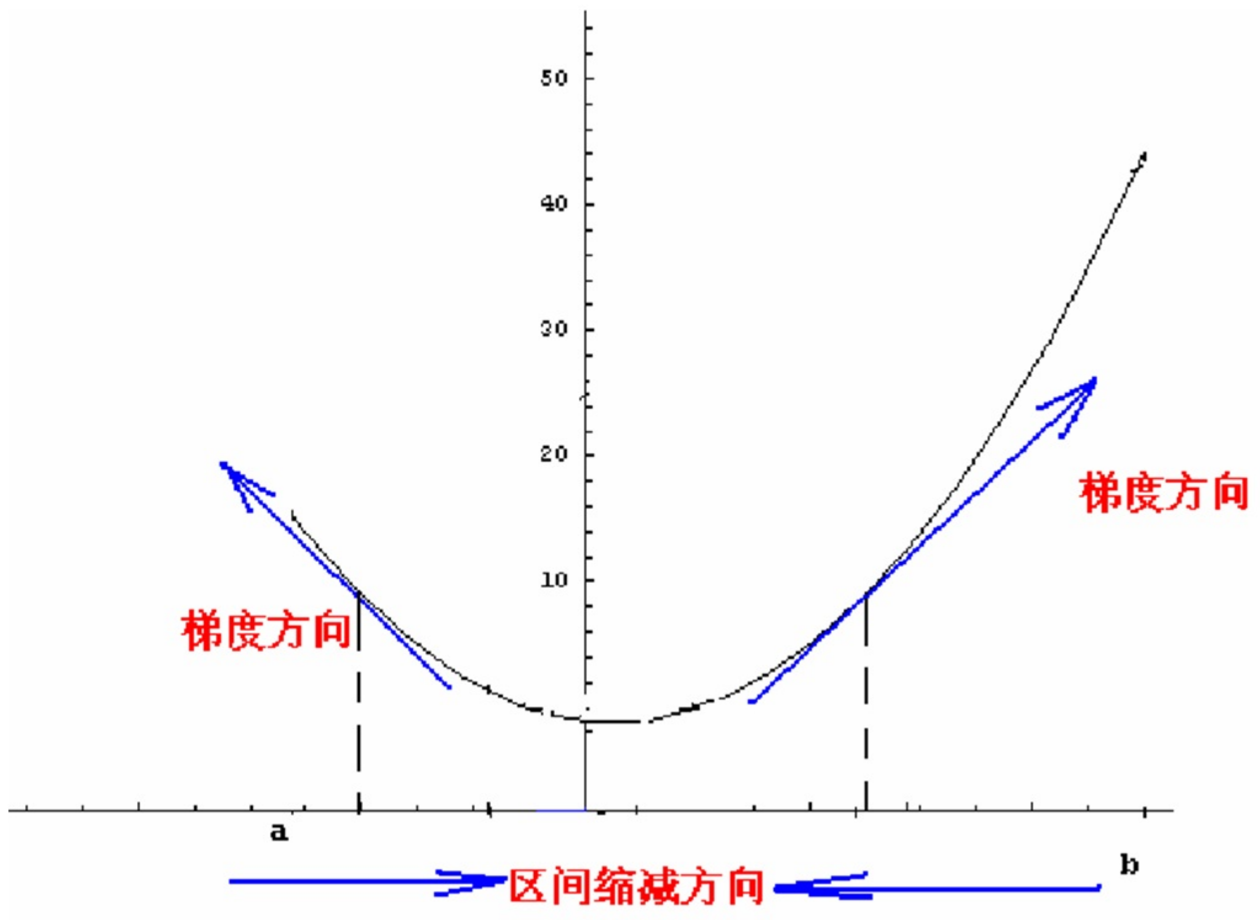
梯度下降法

梯度下降法是一个最优化算法，通常也称为最速下降法。最速下降法是求解无约束优化问题最简单和最古老的方法之一，虽然现在已经不具有实用性，但是许多有效算法都是以它为基础进行改进和修正而得到的。最速下降法是用负梯度方向为搜索方向的，最速下降法越接近目标值，步长越小，前进越慢。

Wiki上的解释为如果目标函数$F(x)$在点a处可微且有定义，那么函数$F(x)$在点a沿着梯度相反的方向$-\nabla F(a)$下降最快。其中,∇为梯度算子，
$\nabla = (\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n})^T$

什么是梯度下降法？

- 梯度下降法，可作为一种求解最小二乘法的方式，它是最优化中比较古老的一种方法
- 梯度下降，设定起始点负梯度方向 (即数值减小的方向) 为搜索方向，寻找最小值。梯度下降法越接近目标值，步长越小，前进越慢.



损失函数 $J(w) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$ 梯度 $\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$

更新规则 $w := w - \eta\frac{\partial J}{\partial w}$

```
class LinearRegressionGD(object):

    def __init__(self, eta=0.001, n_iter=20):
        self.eta = eta  # learning rate 学习速率
        self.n_iter = n_iter  # 迭代次数

    def fit(self, X, y):  # 训练函数
        # self.w_ = np.zeros(1, 1 + X.shape[1])
        self.coef_ = np.zeros(shape=(1, X.shape[1]))  # 代表被训练的系数，初始化为 0
        self.intercept_ = np.zeros(1)
        self.cost_ = []   # 用于保存损失的空list

        for i in range(self.n_iter):
            output = self.net_input(X)  # 计算预测的Y
            errors = y - output
            self.coef_ += self.eta * np.dot(errors.T, X)  # 根据更新规则更新系数，思考一下为什么不是减号？
            self.intercept_ += self.eta * errors.sum()  # 更新 bias，相当于x取常数1
            cost = (errors**2).sum() / 2.0     # 计算损失
            self.cost_.append(cost)  # 记录损失函数的值
        return self

    def net_input(self, X):   # 给定系数和X计算预测的Y
        return np.dot(X, self.coef_.T) + self.intercept_

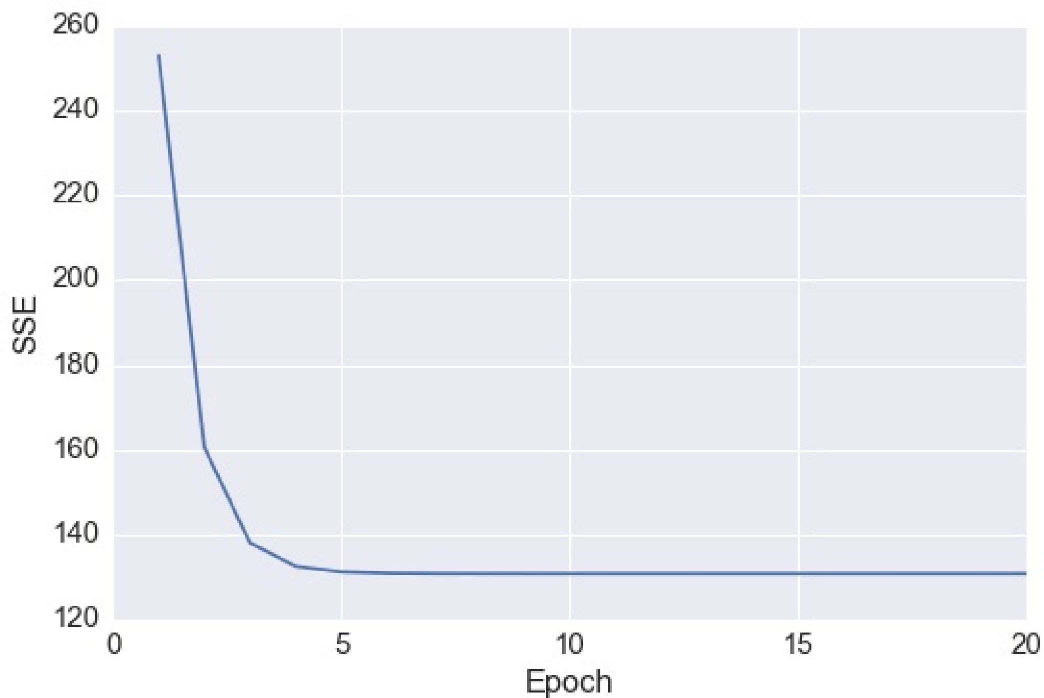
    def predict(self, X):
        return self.net_input(X)
```

```
# RM 作为 explanatory variable
X = df[['RM']].values
y = df[['MEDV']].values
```

```
# standardize
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
sc_y = StandardScaler()
X_std = sc_x.fit_transform(X)
y_std = sc_y.fit_transform(y)
```

```
lr = LinearRegressionGD()
lr.fit(X_std, y_std);  # 喂入数据进行训练
```

```
# cost function
plt.plot(range(1, lr.n_iter+1), lr.cost_)
plt.ylabel('SSE')
plt.xlabel('Epoch')
plt.tight_layout()
```



发现在 epoch 5之后 cost 基本就不能再减小了

```
# 定义一个绘图函数用于展示
def lin_regplot(X, y, model):
    plt.scatter(X, y, c='lightblue')
    plt.plot(X, model.predict(X), color='red', linewidth=2)
    return None
```

```
# 画出预测
lin_regplot(X_std, y_std, lr)
plt.xlabel('Average number of rooms [RM] (standardized)')
plt.ylabel('Price in $1000\'s [MEDV] (standardized)')
plt.tight_layout()
plt.show()
```




```
print('Slope: %.3f' % lr.coef_[0])
print('Intercept: %.3f' % lr.intercept_)
# 直线的斜率及截距
```

```
Slope: 0.695
Intercept: -0.000
```

```
# 预测 RM=5 时，房价为多少
num_rooms_std = sc_x.transform([[5.0]])
price_std = lr.predict(num_rooms_std)
print("Price in $1000's: %.3f" % sc_y.inverse_transform(price_st
d))
```

```
Price in $1000's: 10.840
```

Estimating coefficient of a regression model via scikit-learn

[back to top]

```
from sklearn.linear_model import LinearRegression
```

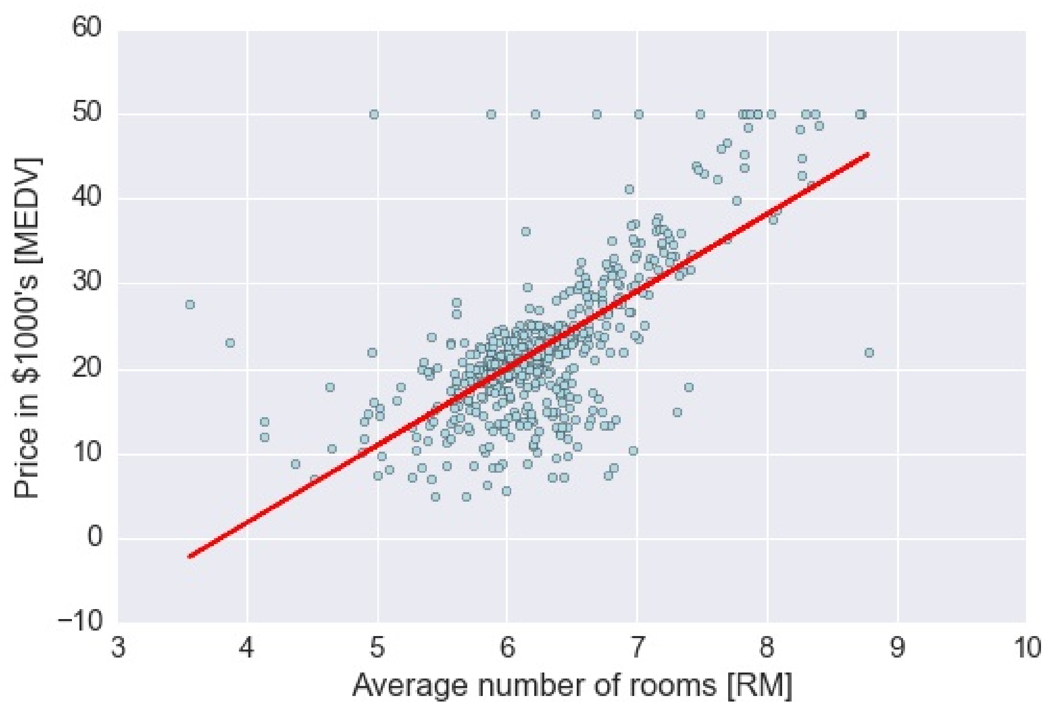
```
slr = LinearRegression()
slr.fit(X_std, y_std)
print('Slope: %.3f' % slr.coef_[0])
print('Intercept: %.3f' % slr.intercept_)
```

```
Slope: 0.695
Intercept: -0.000
```

```
lin_regplot(X_std, y_std, slr)
plt.xlabel('Average number of rooms [RM] (standardized)')
plt.ylabel('Price in $1000\'s [MEDV] (standardized)')
plt.tight_layout()
```




```
# 如果不标准化，直接用原始数据进行回归
slr.fit(X, y)
lin_regplot(X, y, slr)
plt.xlabel('Average number of rooms [RM]')
plt.ylabel('Price in $1000\'s [MEDV]')
plt.tight_layout()
```



结果与使用 gradient descent 的结果接近，思考一下什么时候需要使用标准化？

Fitting a robust regression model using RANSAC

[back to top]

线性回归对 outlier 比较敏感, 而对是否删除 outlier 是需要自己进行判断的. 另一种方法就是 RANdom SAmple Consensus (RANSAC)

大致算法如下:

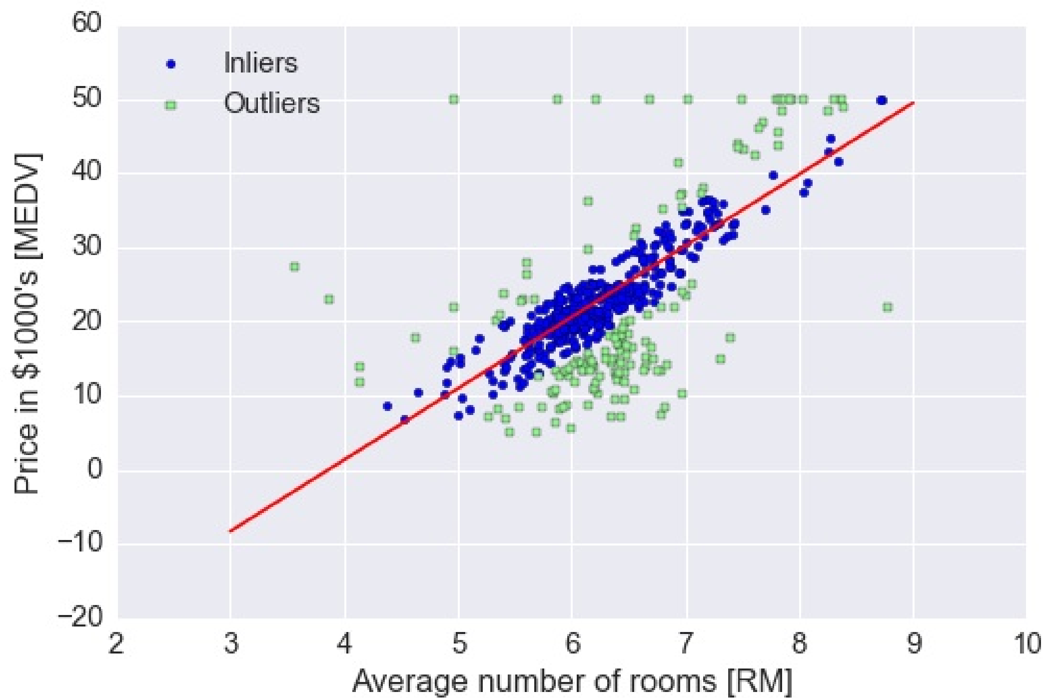
1. Select a random number of samples to be inliers and fit the model.
2. Test all other data points against the fitted model and add those points that fall within a user-given tolerance to the inliers.
3. Refit the model using all inliers.
4. Estimate the error of the fitted model versus the inliers.
5. Terminate the algorithm if the performance meets a certain user-defined threshold or if a fixed number of iterations has been reached; go back to step 1 otherwise.

```
# 使用 sklearn 中已有函数
from sklearn.linear_model import RANSACRegressor
ransac = RANSACRegressor(LinearRegression(),
                         max_trials=100, # max iteration
                         min_samples=50, # min number of randoml
y chosen samples
                         residual_metric=lambda dy: np.sum(np.ab
s(dy), axis=1), # absolute vertical distances to measure
                         residual_threshold=5.0, # allow sample
as inlier within 5 distance units
                         random_state=0)
ransac.fit(X, y)

# 分出 inlier 和 outlier
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

line_X = np.arange(3, 10, 1)
line_y_ransac = ransac.predict(line_X[:, np.newaxis])
plt.scatter(X[inlier_mask], y[inlier_mask], c='blue', marker='o'
, label='Inliers')
plt.scatter(X[outlier_mask], y[outlier_mask], c='lightgreen', ma
rker='s', label='Outliers')
plt.plot(line_X, line_y_ransac, color='red')
plt.xlabel('Average number of rooms [RM]')
plt.ylabel('Price in $1000\'s [MEDV]')
plt.legend(loc='upper left')

plt.tight_layout()
```




```
print('Slope: %.3f' % ransac.estimator_.coef_[0])
print('Intercept: %.3f' % ransac.estimator_.intercept_)
```

```
Slope: 9.621
Intercept: -37.137
```

RANSAC 减少了 outlier 的影响，但对于未知数据的预测能力是否有影响未知.

对比 RANSAC 回归和 OLS 回归

```
from sklearn import datasets

n_samples = 1000
n_outliers = 50


X, y, coef = datasets.make_regression(n_samples=n_samples, n_fea
tures=1,
                                      n_informative=1, noise=10,
                                      coef=True, random_state=0)

# Add outlier data
np.random.seed(0)
```

```
X[:n_outliers] = 3 + 0.5 * np.random.normal(size=(n_outliers, 1)
)
y[:n_outliers] = -3 + 10 * np.random.normal(size=n_outliers)

# Fit line using all data
model = LinearRegression()
model.fit(X, y)

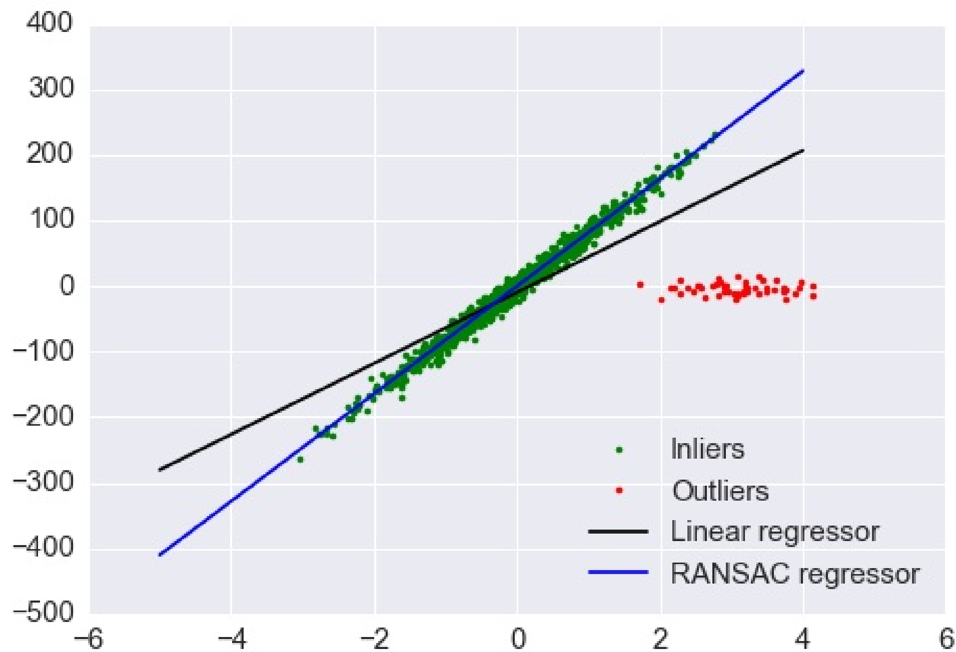
# Robustly fit linear model with RANSAC algorithm
model_ransac = RANSACRegressor(LinearRegression())
model_ransac.fit(X, y)
inlier_mask = model_ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

# Predict data of estimated models
line_X = np.arange(-5, 5)
line_y = model.predict(line_X[:, np.newaxis])
line_y_ransac = model_ransac.predict(line_X[:, np.newaxis])

# Compare estimated coefficients
print("Estimated coefficients (true, normal, RANSAC):")
print(coef, model.coef_, model_ransac.estimator_.coef_)

plt.plot(X[inlier_mask], y[inlier_mask], '.g', label='Inliers')
plt.plot(X[outlier_mask], y[outlier_mask], '.r', label='Outliers'
)
plt.plot(line_X, line_y, '-k', label='Linear regressor')
plt.plot(line_X, line_y_ransac, '-b', label='RANSAC regressor')
plt.legend(loc='lower right');
```

```
Estimated coefficients (true, normal, RANSAC):
(array(82.1903908407869), array([ 54.17236387]), array([ 82.0853
3159]))
```



Evaluating the performance of linear regression models

It is crucial to test the model on data that it hasn't seen during training to obtain an unbiased estimate of its performance.

[back to top]

```
from sklearn.cross_validation import train_test_split

X = df.iloc[:, :-1].values
y = df['MEDV'].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)
# 70% 用于 train, 30%用于 test
```

```
slr = LinearRegression()

slr.fit(X_train, y_train)
y_train_pred = slr.predict(X_train)
y_test_pred = slr.predict(X_test)
```

```
# residual plot, 经常被用来检查回归模型
plt.scatter(y_train_pred,  y_train_pred - y_train, c='blue', marker='o', label='Training data')
plt.scatter(y_test_pred,  y_test_pred - y_test, c='lightgreen', marker='s', label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-10, xmax=50, lw=2, color='red')
plt.xlim([-10, 50])
plt.tight_layout()
# plt.savefig('./figures/slr_residuals.png', dpi=300)
```



如果预测都是正确的, 那么 residual 就是0. 这是理想情况, 实际中, 我们希望 error 是随机分布的.

从上图看, 有部分 error 是离红色线较远的, 可能是 outlier 引起较大的偏差

```
# 另一种评估方法是 Mean Squred Error, MSE, 就是 SSE 的平均值
# R-squre 也是重要的 measurement, 它代表着有多少百分比的数据被模型解释.
越高代表模型拟合越好
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
```

```
MSE train: 19.958, test: 27.196
R^2 train: 0.765, test: 0.673
```

$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})\ R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{MSE}{Var(y)}$

Turning a linear regression model into a curve - Polynomial regression

[back to top]

```
import numpy as np
from
```

```
X = np.array([258.0, 270.0, 294.0,
              320.0, 342.0, 368.0,
              396.0, 446.0, 480.0, 586.0])[:, np.newaxis]

y = np.array([236.4, 234.4, 252.8,
              298.6, 314.2, 342.2,
              360.8, 368.0, 391.2,
              390.8])
```

```
# 添加二次项和截距项

from sklearn.preprocessing import PolynomialFeatures

lr = LinearRegression()
pr = LinearRegression()
quadratic = PolynomialFeatures(degree=2)
X_quad = quadratic.fit_transform(X)
```

```
print(X.shape)
print(X_quad.shape)
```

```
(10, 1)
(10, 3)
```

```
X_quad
```

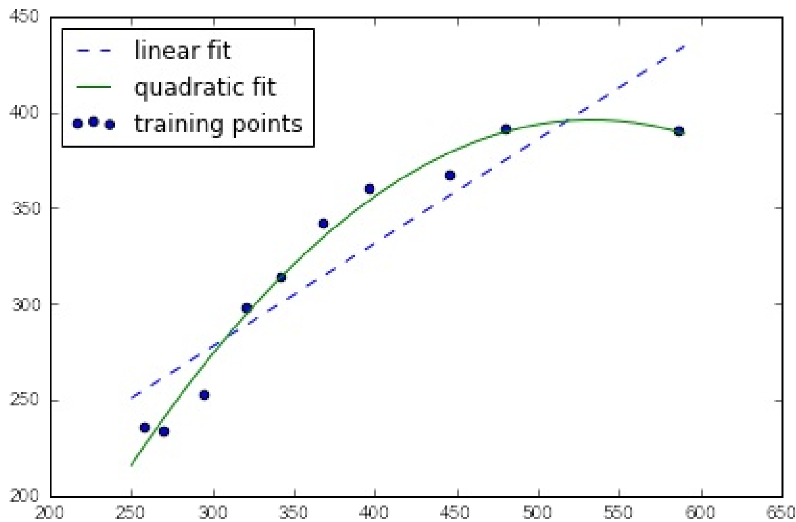
```
array([[  1.00000000e+00,   2.58000000e+02,   6.65640000e+04],
       [  1.00000000e+00,   2.70000000e+02,   7.29000000e+04],
       [  1.00000000e+00,   2.94000000e+02,   8.64360000e+04],
       [  1.00000000e+00,   3.20000000e+02,   1.02400000e+05],
       [  1.00000000e+00,   3.42000000e+02,   1.16964000e+05],
       [  1.00000000e+00,   3.68000000e+02,   1.35424000e+05],
       [  1.00000000e+00,   3.96000000e+02,   1.56816000e+05],
       [  1.00000000e+00,   4.46000000e+02,   1.98916000e+05],
       [  1.00000000e+00,   4.80000000e+02,   2.30400000e+05],
       [  1.00000000e+00,   5.86000000e+02,   3.43396000e+05]])
```

```
# fit linear features
lr.fit(X, y)
X_fit = np.arange(250,600,10)[:, np.newaxis]
y_lin_fit = lr.predict(X_fit)

# fit quadratic features
pr.fit(X_quad, y)
y_quad_fit = pr.predict(quadratic.fit_transform(X_fit))

# plot results
plt.scatter(X, y, label='training points')
plt.plot(X_fit, y_lin_fit, label='linear fit', linestyle='--')
plt.plot(X_fit, y_quad_fit, label='quadratic fit')
plt.legend(loc='upper left')

plt.tight_layout()
```



图上可以发现 quadratic fit比 linear 拟合效果更好

```
y_lin_pred = lr.predict(X)
y_quad_pred = pr.predict(X_quad)
```

```
print('Training MSE linear: %.3f, quadratic: %.3f' % (
        mean_squared_error(y, y_lin_pred),
        mean_squared_error(y, y_quad_pred)))
print('Training  R^2 linear: %.3f, quadratic: %.3f' % (
        r2_score(y, y_lin_pred),
        r2_score(y, y_quad_pred)))
```

```
Training MSE linear: 569.780, quadratic: 61.330
Training  R^2 linear: 0.832, quadratic: 0.982
```

MSE 下降到61, R^2 上升到98%, 说明在这个数据集上 quadratic fit 效果更好

Modeling nonlinear relationships in the Housing dataset

我们会将house prices 与 LSTAT 的 quadratic 及 cubic polynomials fit, 并与 linear fit 对比

back to top

```
X = df[['LSTAT']].values
y = df['MEDV'].values

regr = LinearRegression()

# create quadratic features
quadratic = PolynomialFeatures(degree=2)
cubic = PolynomialFeatures(degree=3)
X_quad = quadratic.fit_transform(X)
X_cubic = cubic.fit_transform(X)

# fit features
X_fit = np.arange(X.min(), X.max(), 1)[:, np.newaxis]

regr = regr.fit(X, y)
y_lin_fit = regr.predict(X_fit)
linear_r2 = r2_score(y, regr.predict(X))

regr = regr.fit(X_quad, y)
y_quad_fit = regr.predict(quadratic.fit_transform(X_fit))
quadratic_r2 = r2_score(y, regr.predict(X_quad))

regr = regr.fit(X_cubic, y)
y_cubic_fit = regr.predict(cubic.fit_transform(X_fit))
cubic_r2 = r2_score(y, regr.predict(X_cubic))



# plot results
plt.scatter(X, y, label='training points', color='lightgray')

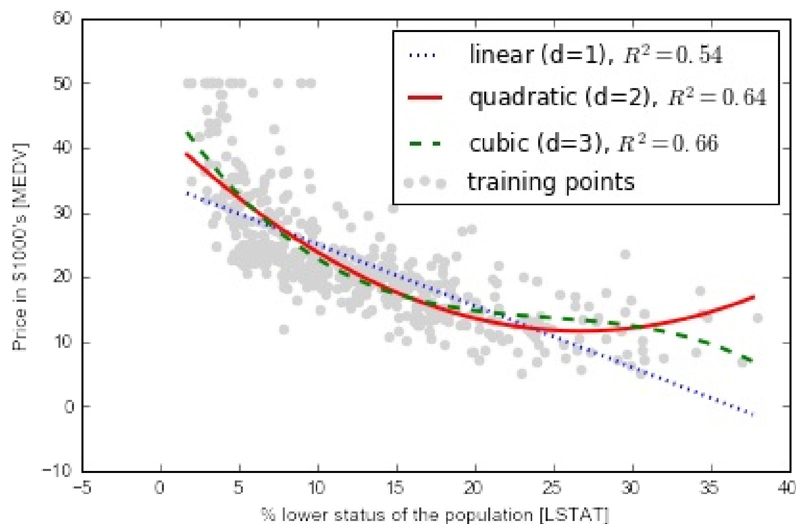
plt.plot(X_fit, y_lin_fit,
         label='linear (d=1), $R^2=%.2f$' % linear_r2,
         color='blue',
         lw=2,
         linestyle=':')
```

```
plt.plot(X_fit, y_quad_fit,
         label='quadratic (d=2), $R^2=%.2f$' % quadratic_r2,
         color='red',
         lw=2,
         linestyle='-')

plt.plot(X_fit, y_cubic_fit,
         label='cubic (d=3), $R^2=%.2f$' % cubic_r2,
         color='green',
         lw=2,
         linestyle='--')

plt.xlabel('% lower status of the population [LSTAT]')
plt.ylabel('Price in $1000\'s [MEDV]')
plt.legend(loc='upper right')

plt.tight_layout()
# plt.savefig('./figures/polyhouse_example.png', dpi=300)
```



Transforming the dataset by log: 为什么要这样做？是因为有画图探索的启示？

```
X = df[['LSTAT']].values
y = df['MEDV'].values

# transform features
X_log = np.log(X)
y_sqrt = np.sqrt(y)

# fit features
X_fit = np.arange(X_log.min()-1, X_log.max()+1, 1)[:, np.newaxis]

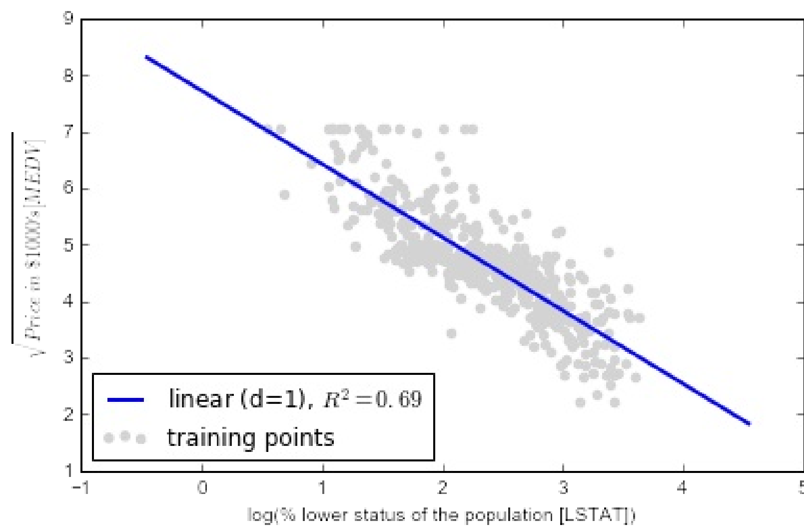
regr = regr.fit(X_log, y_sqrt)
y_lin_fit = regr.predict(X_fit)
linear_r2 = r2_score(y_sqrt, regr.predict(X_log))

# plot results
plt.scatter(X_log, y_sqrt, label='training points', color='lightgray')

plt.plot(X_fit, y_lin_fit,
         label='linear (d=1), $R^2=%.2f$' % linear_r2,
         color='blue',
         lw=2)

plt.xlabel('log(% lower status of the population [LSTAT])')
plt.ylabel('$\sqrt{Price \; in \; \$1000\'s [MEDV]}$')
plt.legend(loc='lower left')

plt.tight_layout()
# plt.savefig('./figures/transform_example.png', dpi=300)
plt.show()
```



经过 log 变换后，线性拟合效果已经不错, 比单纯 polynomial fit 更好

练习：用房价数据的其它自变量一起做一个多元模型看看**R2**有没有改善

Sections

什么是感知机分类

最简单形式的前馈神经网络，是一种二元线性分类器, 把矩阵上的输入 x （实数值向量）映射到输出值 $f(x)$ 上（一个二元的值）。

$$f(x) = \begin{cases} +1 & \text{if } w \cdot x + b > 0 \\ -1 & \text{else} \end{cases}$$

学习算法

我们首先定义一些变量：

- $x(j)$ 表示n维输入向量中的第j项
- $w(j)$ 表示权重向量的第j项
- $f(x)$ 表示神经元接受输入 x 产生的输出
- α 是一个常数，符合 $0 < \alpha \leq 1$ （接受率）
- 更进一步，为了简便我们假定偏置量 b 等于0。因为一个额外的维度 $n+1$ 维，可以用 $x(n+1)=1$ 的形式加到输入向量，这样我们就可以用 $w(n+1)$ 代替偏置量。

感知器的学习通过对所有训练实例进行多次的迭代进行更新的方式来建模。

令 $D_m = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ 表示一个有 m 个训练实例的训练集。

每次迭代权重向量以如下方式更新： 对于每个 $D_m = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ 中的每个 (x, y) 对， $w(j) := w(j) + \alpha(y - f(x))x(j) \quad (j = 1, \ldots, n)$

注意这意味着，仅当针对给定训练实例 (x, y) 产生的输出值 $f(x)$ 与预期的输出值 y 不同时，权重向量才会发生改变。

如果存在一个正的常数 γ 和权重向量 w ，对所有的 i 满足 $y_i \cdot (\langle w, x_i \rangle + b) > \gamma$ ，训练集 D_m 就被叫做线性分隔。 然而，如果训练集不是线性分隔的，那么这个算法则不能确保会收敛。

Implementing a perceptron learning algorithm in Python

[back to top]

```
import numpy as np


class Perceptron(object):
    """Perceptron classifier.

    Parameters
```

```
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every epoch.

    """
    def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta
        self.n_iter = n_iter  # the number of epochs

    def fit(self, X, y):
        """Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of s
amples and
            n_features is the number of features.
        y : array-like, shape = [n_samples]
            Target values.

        Returns
        -------
        self : object

        """
        self.w_ = np.zeros(1 + X.shape[1])  # weights, 初始值0
        self.errors_ = []

        # 对每个 sample 循环更新
        for _ in range(self.n_iter):
```

```
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
#learning rate*error
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)  # 错误的分类结果
        return self

    def net_input(self, X):
        """Calculate net input w*x"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

Training a perceptron model on the Iris dataset

这里只考虑两种花 Setosa 和 Versicolor , 以及两种特征 sepal length 和 petal length.

但是 Perceptron Model 可以解决多类别分类问题, 参考 one-vs-all

[back to top]

Reading-in the Iris data

```
import pandas as pd

df = pd.read_csv('data/iris.csv', header=None)
df.tail()
```

	0	1	2	3	4
146	6.7	3	5.2	2.3	virginica
147	6.3	2.5	5	1.9	virginica
148	6.5	3	5.2	2	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3	5.1	1.8	virginica

Plotting the Iris data

```
# 将两个分类先可视化
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

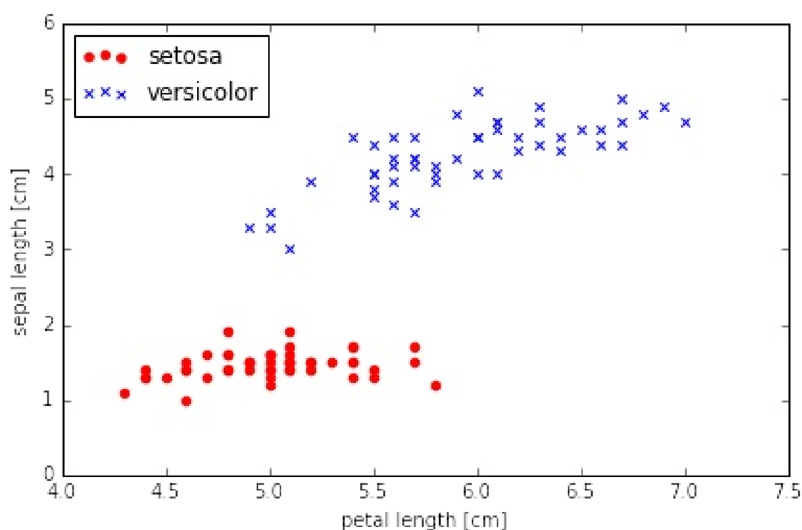
# select setosa and versicolor
# 两种各选择50个, 把类别改为 -1 和 1, 方便画图
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

# extract sepal length and petal length
X = df.iloc[0:100, [0, 2]].values

# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')

plt.xlabel('petal length [cm]')
plt.ylabel('sepal length [cm]')
plt.legend(loc='upper left')

plt.tight_layout()
# plt.savefig('./iris_1.png', dpi=300)
```



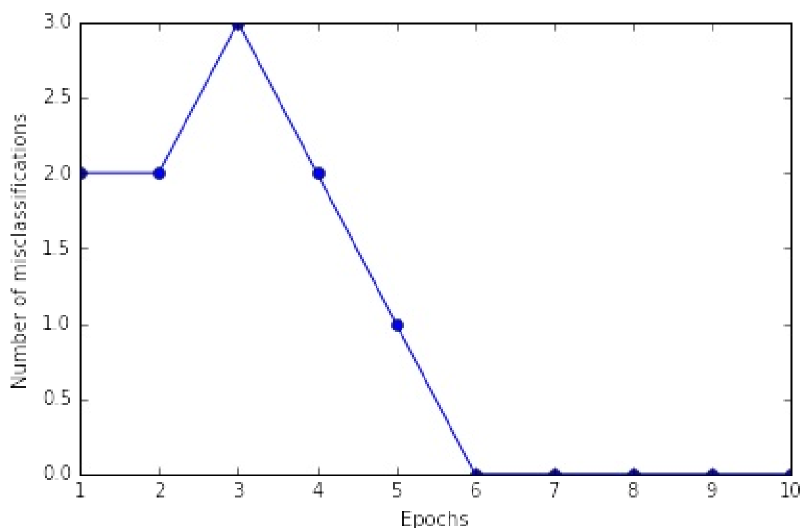
Training the perceptron model

```
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
ppn.errors_
```

```
[2, 2, 3, 2, 1, 0, 0, 0, 0, 0]
```

```
# error 画图, 检查是否 error 趋近于0 在多次 loop 更新后
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o'
)
plt.xlabel('Epochs')
plt.ylabel('Number of misclassifications')

plt.tight_layout()
# plt.savefig('./perceptron_1.png', dpi=300)
```



结果 error 的确最后为 0, 证明是 convergent 的, 且分类效果应该说是非常准确了

A function for plotting decision regions

这个函数用于画决策边界

```
from matplotlib.colors import ListedColormap
# Colormap object generated from a list of colors.

def plot_decision_regions(X, y, classifier, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface 确定横纵轴边界
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1  # 最小
-1, 最大+1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1

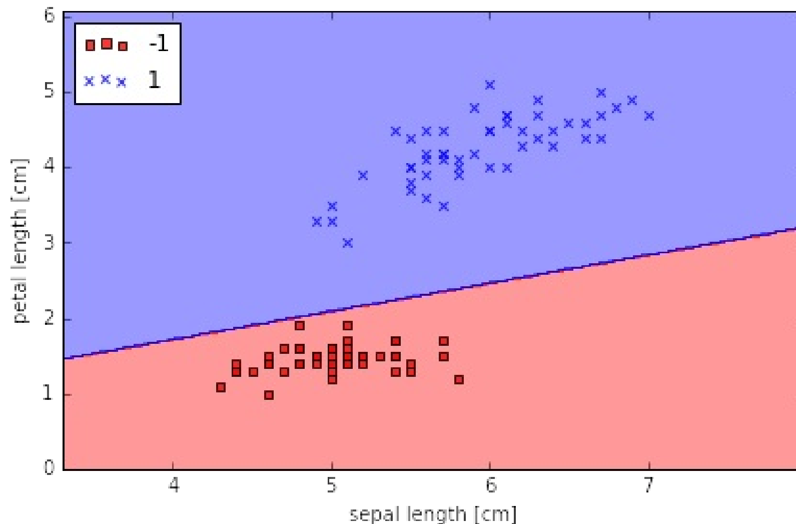
    # create a pair of grid arrays
    # flatten the grid arrays then predict
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution)
,
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).
T)
    Z = Z.reshape(xx1.shape)

    # maps the different decision regions to different colors
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)
```

```
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')

plt.tight_layout()
# plt.savefig('./perceptron_2.png', dpi=300)
```



虽然 Perceptron Model 在上面 Iris 例子里表现得很好，但在其他问题上却不一定表现得好。 Frank Rosenblatt 从数学上证明了，在线性可分的数据里，Perceptron 的学习规则会 converge，但在线性不可分的情况下，却无法 converge

Adaptive linear neurons and the convergence of learning (Adaline)

[back to top]

Implementing an adaptive linear neuron in Python

[back to top]

ADAptive LInear NEuron classifier 也是一个单层神经网络. 它的重点就是定义及最优化 cost function, 对于理解更高层次更难的机器学习分类模型是非常好的入门.

它与 Perceptron 不同的地方在于更新 weights 时是用的 linear activation function, 而不是unit step function.

Adaline 中这个 linear activation function 输出等于输入, $\varphi(w^T x) = w^T x$.

然后 activation 后会有一个 quantizer 用来学习更新 weights

定义 cost function 为 SSE: Sum of Squared Errors

$$J(w) = \frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2$$

这个 function 是可导的, 并且是 convex 的, 可以进行最优化, 使用 gradient descent 算法.

```
class AdalineGD(object):
    """ADAptive LInear NEuron classifier.

    Parameters
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every epoch.

    """
    def __init__(self, eta=0.01, n_iter=50):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
```

```
        """ Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of s
amples and
            n_features is the number of features.
        y : array-like, shape = [n_samples]
            Target values.

        Returns
        -------
        self : object

        """
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        # gradient descent
        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost) # cost list, to check algori
thm convergence
        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        """Compute linear activation"""
        return self.net_input(X)

    def predict(self, X):
        """Return class label after unit step"""
```

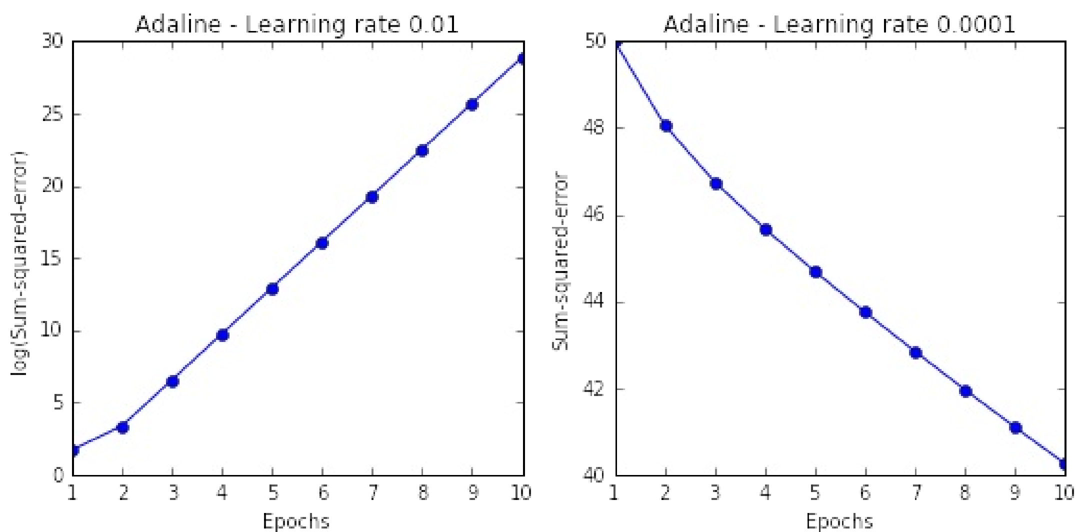
```
        return np.where(self.activation(X) >= 0.0, 1, -1)
```

```
# 测试两种 learning rate, 0.01 和 0.0001
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

ada1 = AdalineGD(n_iter=10, eta=0.01).fit(X, y)
ax[0].plot(range(1, len(ada1.cost_) + 1), np.log10(ada1.cost_),
marker='o')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('log(Sum-squared-error)')
ax[0].set_title('Adaline - Learning rate 0.01')

ada2 = AdalineGD(n_iter=10, eta=0.0001).fit(X, y)
ax[1].plot(range(1, len(ada2.cost_) + 1), ada2.cost_, marker='o'
)
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Sum-squared-error')
ax[1].set_title('Adaline - Learning rate 0.0001')

plt.tight_layout()
# plt.savefig('./adaline_1.png', dpi=300)
```



左图显示 learning rate 太大, error 没有变小, 反而变大了.

右图显示 learning rate 太小, error 变化速度太小

Standardizing features and re-training adaline

$x'_j = \frac{x_j - \mu_j}{\sigma_j}$

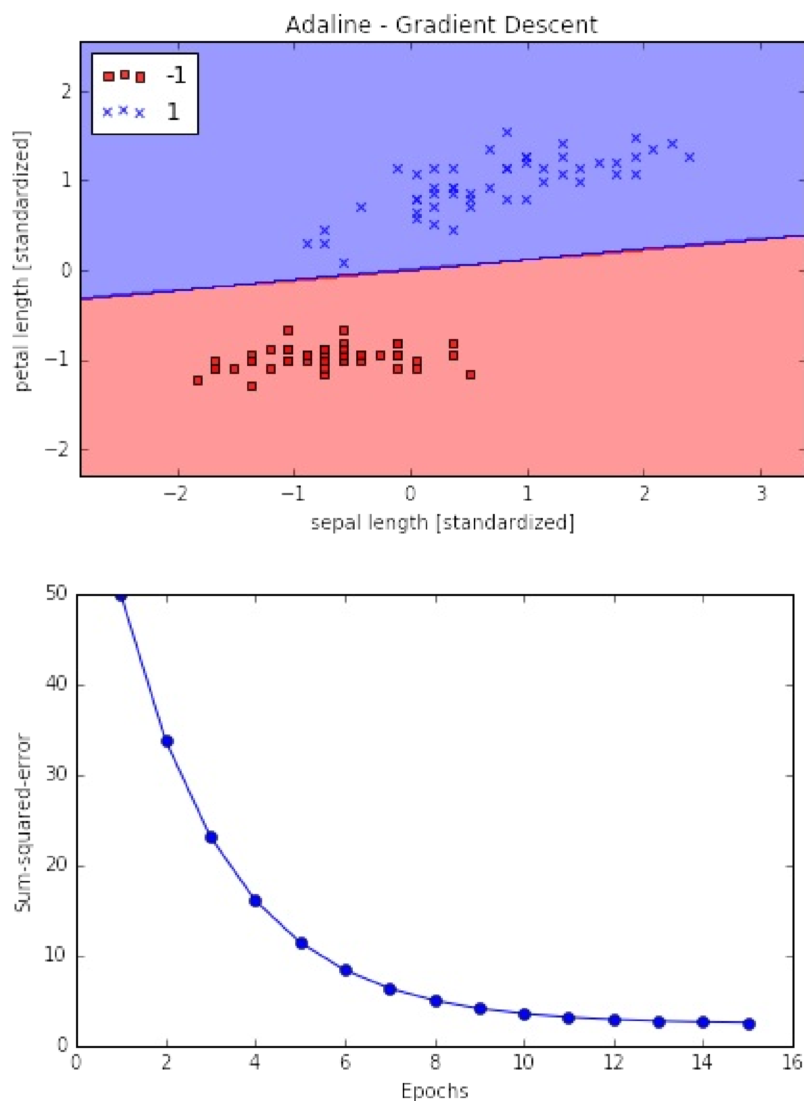
```
# standardize features
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()
```

```
ada = AdalineGD(n_iter=15, eta=0.01)
ada.fit(X_std, y)

plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Gradient Descent')
plt.xlabel('sepal length [standardized]')
plt.ylabel('petal length [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./adaline_2.png', dpi=300)
plt.show()

plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Sum-squared-error')

plt.tight_layout()
# plt.savefig('./adaline_3.png', dpi=300)
```



分类效果不错, error 最终接近于0 值得注意的是，虽然我们的分类全部正确，但 error 也不等于0

```
ada.w_ #weights
```

```
array([  1.36557432e-16,  -1.26256159e-01,   1.10479201e+00])
```

Large scale machine learning and stochastic gradient descent

Stochastic gradient descent 随机梯度下降 比一般的梯度下降更有优势, 因为每一步计算的 cost 更小, 每一步更新都是随机取其中一小步更新就可.

batch gradient descent 一次更新需要计算一遍整个数据集
$\Delta w = \eta \sum_i (y^{(i)} - \phi(z^{(i)}))x^{(i)}$

stochastic gradient descent 一次更新只需计算一个数据点
$\Delta w = \eta (y^{(i)} - \phi(z^{(i)}))x^{(i)}$

```
class AdalineSGD(object):
    """ADAptive LInear NEuron classifier.

    Parameters
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every epoch.
    shuffle : bool (default: True)
        Shuffles training data every epoch if True to prevent cy
cles.
    random_state : int (default: None)
        Set random state for shuffling and initializing the weig
hts.

    """
    def __init__(self, eta=0.01, n_iter=10, shuffle=True, random
_state=None):
        self.eta = eta
        self.n_iter = n_iter
        self.w_initialized = False
        self.shuffle = shuffle
        if random_state:  # allow the specication of a random se
```

```
ed for consistency
            np.random.seed(random_state)

    def fit(self, X, y):
        """ Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of s
amples and
            n_features is the number of features.
        y : array-like, shape = [n_samples]
            Target values.

        Returns
        -------
        self : object

        """
        self._initialize_weights(X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            if self.shuffle:
                X, y = self._shuffle(X, y)
            cost = []
            for xi, target in zip(X, y):
                cost.append(self._update_weights(xi, target))
            avg_cost = sum(cost) / len(y)
            self.cost_.append(avg_cost)
        return self

    # 在每个 epoch 前是否 shuffle data
    def _shuffle(self, X, y):
        """Shuffle training data"""
        r = np.random.permutation(len(y))
        return X[r], y[r]

    def _initialize_weights(self, m):
        """Initialize weights to zeros"""
```

```
            self.w_ = np.zeros(1 + m)
            self.w_initialized = True

        #stochastic gradient descent
        def _update_weights(self, xi, target):
            """Apply Adaline learning rule to update the weights"""
            output = self.net_input(xi)
            error = (target - output)
            self.w_[1:] += self.eta * xi.dot(error) # 仅一个 error 相乘

            self.w_[0] += self.eta * error # 仅仅是一个 error, 而非 sum

            cost = 0.5 * error**2
            return cost

        def net_input(self, X):
            """Calculate net input"""
            return np.dot(X, self.w_[1:]) + self.w_[0]

        def activation(self, X):
            """Compute linear activation"""
            return self.net_input(X)

        def predict(self, X):
            """Return class label after unit step"""
            return np.where(self.activation(X) >= 0.0, 1, -1)
```

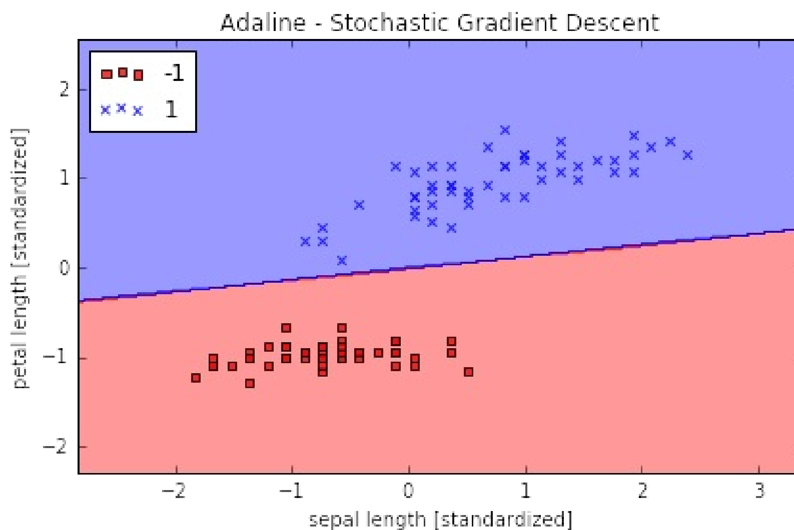
```
# plot result
ada = AdalineSGD(n_iter=15, eta=0.01, random_state=1)
ada.fit(X_std, y)

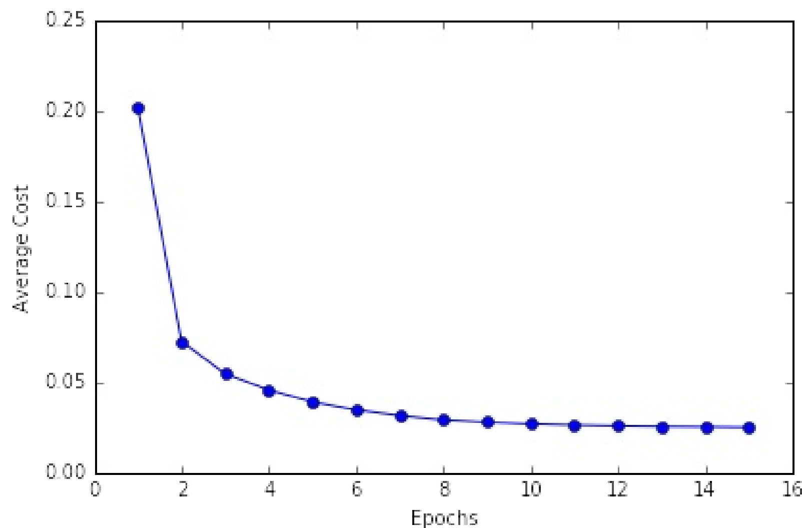
plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Stochastic Gradient Descent')
plt.xlabel('sepal length [standardized]')
plt.ylabel('petal length [standardized]')
plt.legend(loc='upper left')

plt.tight_layout()
#plt.savefig('./adaline_4.png', dpi=300)
plt.show()

plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Average Cost')

plt.tight_layout()
# plt.savefig('./adaline_5.png', dpi=300)
```





对比可以发现，Stochastic gradient descent 的学习速率比 Batch gradient descnent 的高很多

Implementing logistic regression in Python

[back to top]

logistic regression: powerful algorithm for linear and binary classication problems

odds ratio: the odds in favor of a particular event. $\frac{p}{(1-p)}$, if p stands for the probability of the positive event, we want to predict. $logit(p) = log\frac{p}{1-p}$

and $p(y=1|x)$ is conditional probability that a particular sample belongs to class 1 given its features x. Therefore, the logit is

$$logit(p(y=1|x)) = w_0x_0 + w_1x_1 + ... + w_mx_m = \sum_{i=0}^{n} w_ix_i$$

we want to know the probability, which is inverse of logit function, we call it logistic funcition, or **sigmoid function**. $\phi(z) = \frac{1}{1+e^{-z}}$

output of sigmoid function is as the probability of particular sample belonging to class 1

Plot sigmoid function:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

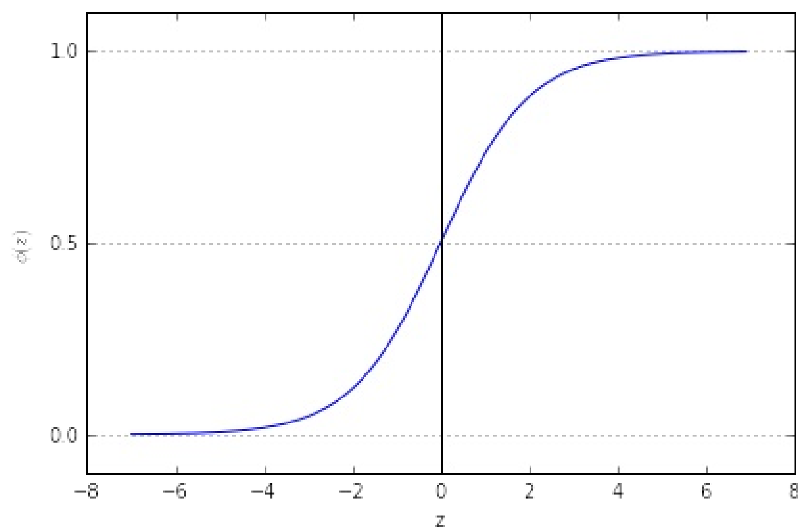
def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))

z = np.arange(-7, 7, 0.1)
phi_z = sigmoid(z)

plt.plot(z, phi_z)
plt.axvline(0.0, color='k')
plt.ylim(-0.1, 1.1)
plt.xlabel('z')
plt.ylabel('$\phi (z)$')

# y axis ticks and gridline
plt.yticks([0.0, 0.5, 1.0])
ax = plt.gca()
ax.yaxis.grid(True)

plt.tight_layout()
# plt.savefig('./figures/sigmoid.png', dpi=300)
```



when $\phi(z)$ approached 1 if z→∞, goes to 1 if z→$-\infty$

Plot cost function:

use log-likelihood function to redefine cost function

then $$J(\phi(z), y; w) = \begin{cases} -log(\phi(z)) & \text{if } y = 1 \\ -log(1 - \phi(z)) & \text{if } y = 0 \end{cases}$$

```
def cost_1(z):
    return - np.log(sigmoid(z))

def cost_0(z):
    return - np.log(1 - sigmoid(z))

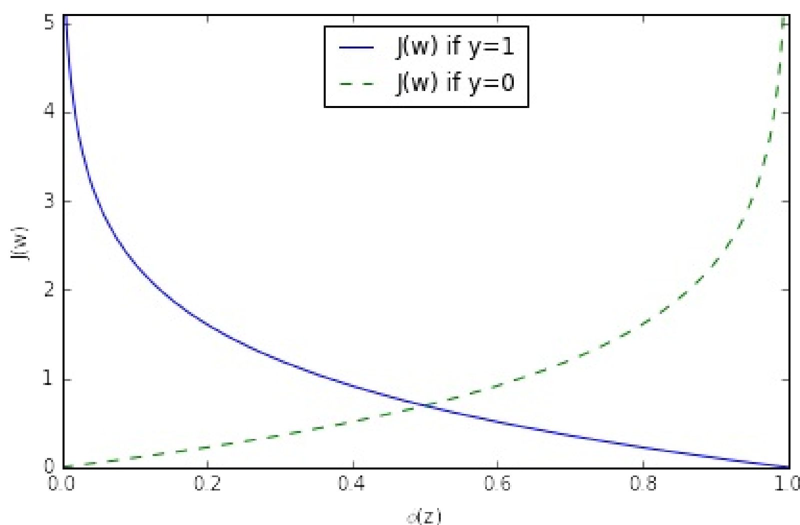
z = np.arange(-10, 10, 0.1)
phi_z = sigmoid(z)

c1 = [cost_1(x) for x in z]
plt.plot(phi_z, c1, label='J(w) if y=1')

c0 = [cost_0(x) for x in z]
plt.plot(phi_z, c0, linestyle='--', label='J(w) if y=0')

plt.ylim(0.0, 5.1)
plt.xlim([0, 1])
plt.xlabel('$\phi$(z)')
plt.ylabel('J(w)')
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('./figures/log_cost.png', dpi=300)

# this illustrates the cost for the classification of a single-s
ample instance for diff values of phi(z)
```



cost 趋近于0 如果正确预测 class.

Implement in Python

The following implementation is similar to the Adaline implementation except that we replace the sum of squared errors cost function with the logistic cost function

$$J(\mathbf{w}) = \sum_{i=1}^{m} -y^{(i)} log\left(\phi\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right) log\left(1 - \phi\left(z^{(i)}\right)\right).$$

```
class LogisticRegression(object):
    """LogisticRegression classifier.

    Parameters
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    cost_ : list
        Cost in every epoch.

    """
    def __init__(self, eta=0.01, n_iter=50):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        """ Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of s
amples and
            n_features is the number of features.
        y : array-like, shape = [n_samples]
```

```
            Target values.

        Returns
        -------
        self : object

        """
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            y_val = self.activation(X)
            errors = (y - y_val)
            neg_grad = X.T.dot(errors)
            self.w_[1:] += self.eta * neg_grad
            self.w_[0] += self.eta * errors.sum()
            self.cost_.append(self._logit_cost(y, self.activatio
n(X)))
        return self

    def _logit_cost(self, y, y_val):
        logit = -y.dot(np.log(y_val)) - ((1 - y).dot(np.log(1 -
y_val)))
        return logit

    def _sigmoid(self, z):
        return 1.0 / (1.0 + np.exp(-z))

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        """ Activate the logistic neuron"""
        z = self.net_input(X)
        return self._sigmoid(z)

    def predict_proba(self, X):
        """
        Predict class probabilities for X.
```

```
        Parameters
        ----------
        X : {array-like, sparse matrix}, shape = [n_samples, n_f
eatures]
            Training vectors, where n_samples is the number of s
amples and
            n_features is the number of features.

        Returns
        ----------
          Class 1 probability : float

        """
        return activation(X)

    def predict(self, X):
        """
        Predict class labels for X.

        Parameters
        ----------
        X : {array-like, sparse matrix}, shape = [n_samples, n_f
eatures]
            Training vectors, where n_samples is the number of s
amples and
            n_features is the number of features.

        Returns
        ----------
        class : int
            Predicted class label.

        """
        # equivalent to np.where(self.activation(X) >= 0.5, 1, 0)

        return np.where(self.net_input(X) >= 0.0, 1, 0)
```

```
y[y == -1] = 0  # 将负性标签编码为 0
y
```

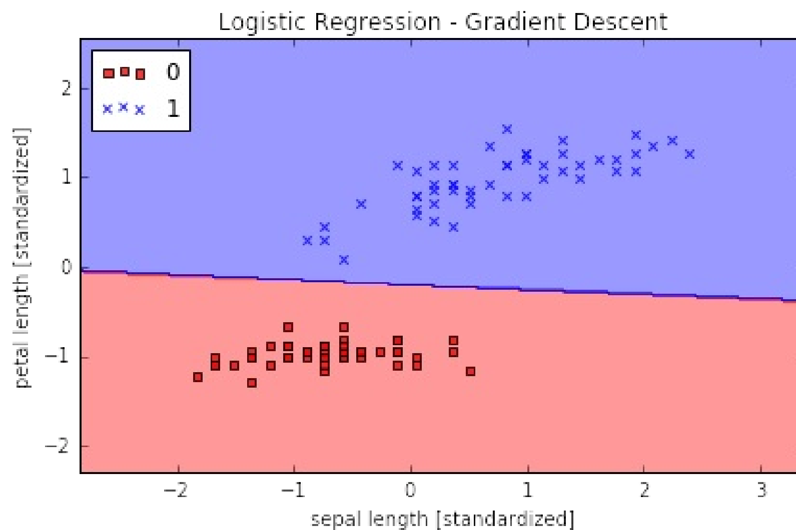
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1])
```

```
lr = LogisticRegression(n_iter=500, eta=0.02).fit(X_std, y)
plt.plot(range(1, len(lr.cost_) + 1), np.log10(lr.cost_))
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Logistic Regression - Learning rate 0.02')

plt.tight_layout()
```



```
plot_decision_regions(X_std, y, classifier=lr)
plt.title('Logistic Regression - Gradient Descent')
plt.xlabel('sepal length [standardized]')
plt.ylabel('petal length [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
```



Classification with scikit-learn

[back to top]

Loading and preprocessing the data

[back to top]

Loading the **Iris dataset** from scikit-learn. Here, the third column represents the petal length, and the fourth column the petal width of the flower samples. The classes are already converted to integer labels where 0=Iris-Setosa, 1=Iris-Versicolor, 2=Iris-Virginica.

```
from sklearn import datasets
import numpy as np

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

print('Class labels:', np.unique(y))
print(iris.target_names)
```

```
('Class labels:', array([0, 1, 2]))
['setosa' 'versicolor' 'virginica']
```

Splitting data into 70% training and 30% test data:

```
from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.3, random_state=0)
```

Standardizing the features:

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train) # standardize by mean & std
X_test_std = sc.transform(X_test)
```

Other Available Data

[back to top]

Scikit-learn makes available a host of datasets for testing learning algorithms. They come in three flavors:

- **Packaged Data:** these small datasets are packaged with the scikit-learn installation, and can be downloaded using the tools in `sklearn.datasets.load_*`
- **Downloadable Data:** these larger datasets are available for download, and scikit-learn includes tools which streamline this process. These tools can be found in `sklearn.datasets.fetch_*`
- **Generated Data:** there are several datasets which are generated from models based on a random seed. These are available in the `sklearn.datasets.make_*`

You can explore the available dataset loaders, fetchers, and generators using IPython's tab-completion functionality. After importing the `datasets` submodule from `sklearn`, type

```
datasets.load_<TAB>
```

or

```
datasets.fetch_<TAB>
```

or

```
datasets.make_<TAB>
```

to see a list of available functions.

The data downloaded using the `fetch_` scripts are stored locally, within a subdirectory of your home directory. You can use the following to determine where it is:

```
from sklearn.datasets import get_data_home
get_data_home()
```

Training a perceptron via scikit-learn

[back to top]

```
from sklearn.linear_model import Perceptron
# sklearn 中有封装好的 Perceptron 函数
ppn = Perceptron(n_iter=40, eta0=0.1, random_state=0)
ppn.fit(X_train_std, y_train)
```

```
Perceptron(alpha=0.0001, class_weight=None, eta0=0.1, fit_interc
ept=True,
      n_iter=40, n_jobs=1, penalty=None, random_state=0, shuffle
=True,
      verbose=0, warm_start=False)
```

```
y_test.shape
```

```
(45,)
```

```
y_pred = ppn.predict(X_test_std) # predict
print('Misclassified samples: %d' % (y_test != y_pred).sum()) #
错误个数
```

```
Misclassified samples: 4
```

```
from sklearn.metrics import accuracy_score

print('Accuracy: %.2f' % accuracy_score(y_test, y_pred)) # 91%
的准确率
```

```
Accuracy: 0.91
```

```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
%matplotlib inline

# 重新定义画决策边界函数，使得能区分训练数据和测试数据
def plot_decision_regions(X, y, classifier, test_idx=None, resol
ution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution)
,
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).
T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot all samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)

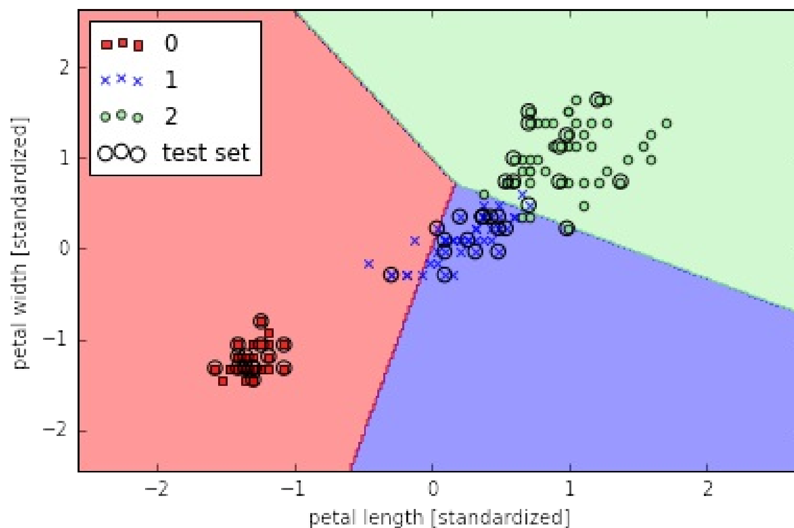
    # highlight test samples
    if test_idx:
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1], c='',
                alpha=1.0, linewidth=1, marker='o',
                s=55, label='test set')
```

Training a perceptron model using the standardized training data:

```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

plot_decision_regions(X=X_combined_std, y=y_combined,
                      classifier=ppn, test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')

plt.tight_layout()
# plt.savefig('./figures/iris_perceptron_scikit.png', dpi=300)
# 这次是3个分类一起
```



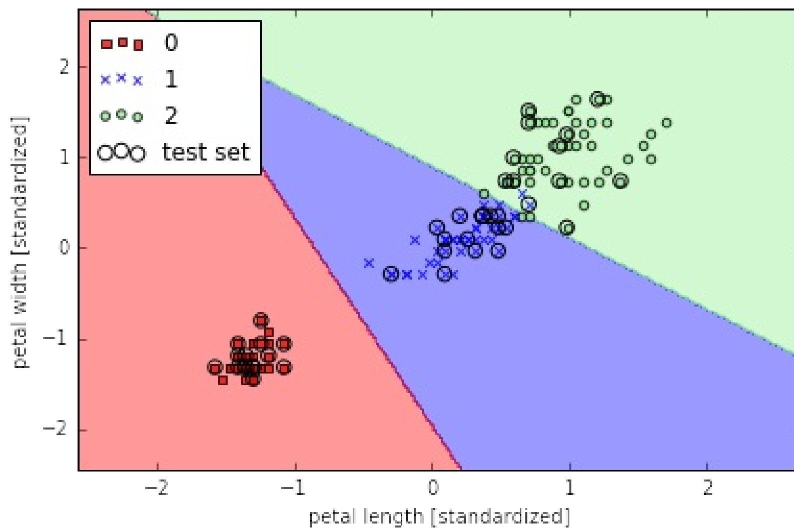
Peceptron 模型对于并不是完全线性隔离的 dataset 不能 converge, 所以实际应用中并不多用.

Modeling class probabilities via logistic regression

[back to top]

```
# use Logistic Regression
from sklearn.linear_model import LogisticRegression
# C parameter 是什么呢?
lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=lr, test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/logistic_regression.png', dpi=300)
```




```
lr.predict_proba(X_test_std[0,:].reshape(1,-1)) # predict probab
ility
```

```
array([[  2.05743774e-11,   6.31620264e-02,   9.36837974e-01]])
```

Regularization path

解决 overfitting: 模型拟合的过好, 以致于没有一般性, 预测新的样本的结果就会很差

一般过拟合的模型会有 high variance

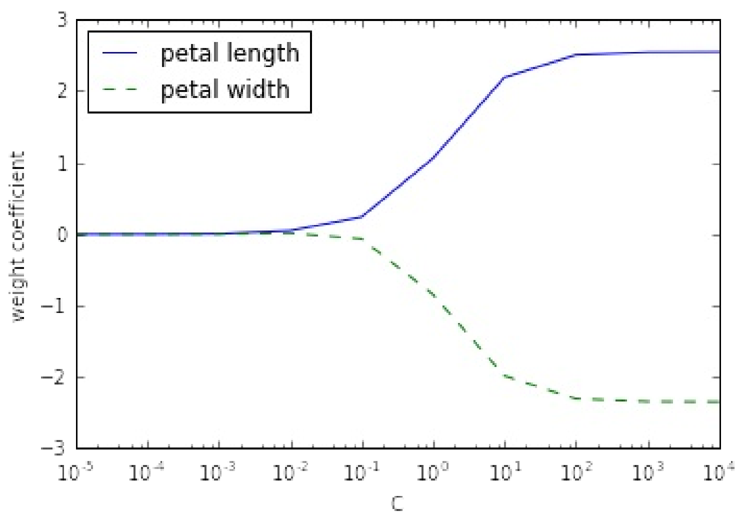
最常用的解决方法就叫做 L2 regulatization $\frac{\lambda}{2}\|w\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$

其中λ 就是 regularization parameter, 可以用来控制拟合训练数据的好坏, 而 $C = \frac{1}{\lambda}$ 就是前面提到过的 parameter

```
weights, params = [], []
for c in np.arange(-5, 5):
    lr = LogisticRegression(C=10**c, random_state=0)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)
plt.plot(params, weights[:, 0],
         label='petal length')
plt.plot(params, weights[:, 1], linestyle='--',
         label='petal width')
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.legend(loc='upper left')
plt.xscale('log')
# plt.savefig('./figures/regression_path.png', dpi=300)

# C 减小的话, 就是增加 regularization
```



Logistic regression with regularization

```
class LogitGD(object):
    """Logistic Regression classifier.

    Parameters
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every epoch.

    """
    def __init__(self, eta=0.01, lamb = 0.01, n_iter=50):
        self.eta = eta
        self.n_iter = n_iter
        self.lamb = lamb

    def fit(self, X, y):
        """ Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of s
amples and
            n_features is the number of features.
        y : array-like, shape = [n_samples]
            Target values.
```

```
        Returns
        -------
        self : object

        """
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []


        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors) - self.lam
b* self.w_[1:]
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum() / 2.0 + self.lamb* np.sum(s
elf.w_[1:]**2)
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def sigmoid(z):
        return 1.0 / (1.0 + np.exp(-z))

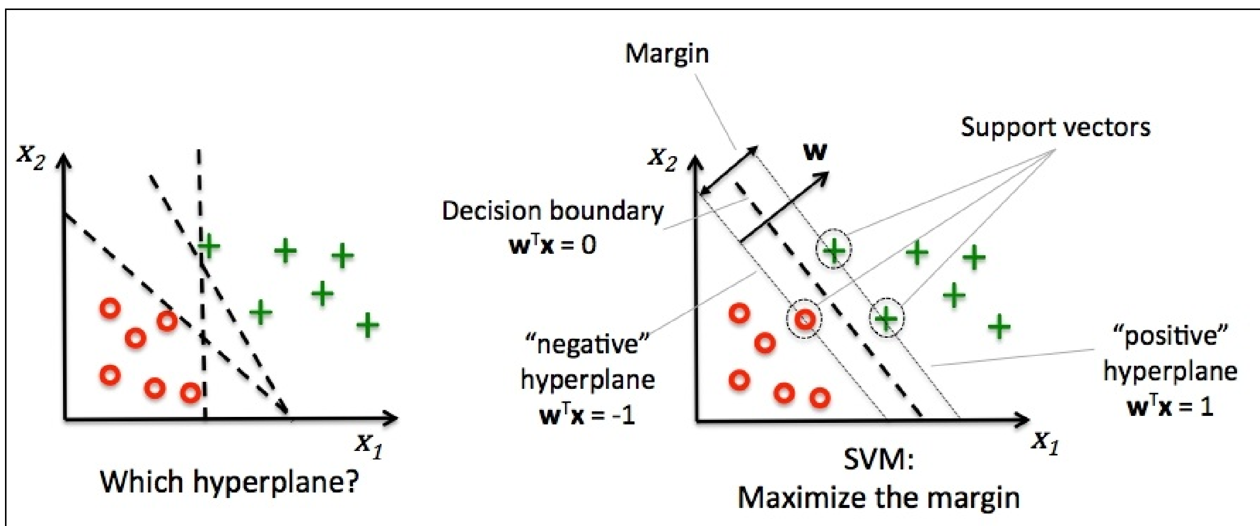
    def activation(self, X):
        """Compute linear activation"""
        return sigmoid(self.net_input(X))

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.activation(X) >= 0.5, 1, -1)
```

其它的分类器简介

Maximum margin classification with support vector machines

目的是 maximize the **margin**, margin 是分离决策边界与离之最近的训练样本之间的距离.

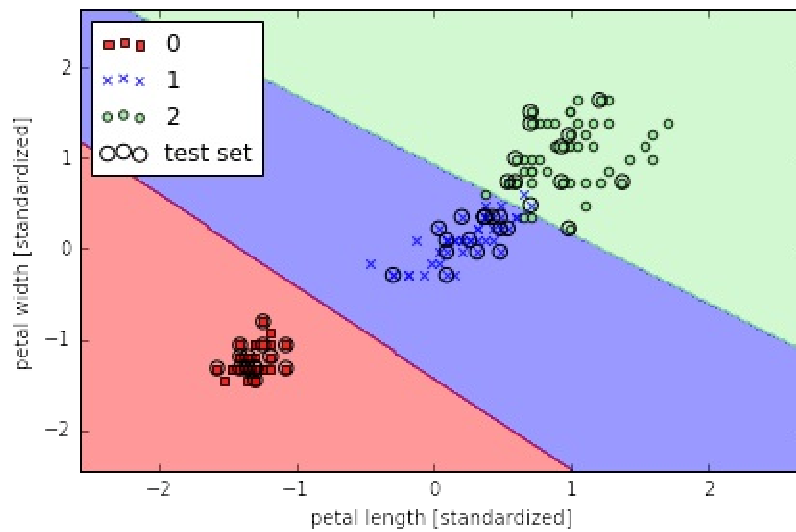


[back to top]

```
# train SVC
from sklearn.svm import SVC

svm = SVC(kernel='linear', C=1.0, random_state=0)
svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=svm, test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/support_vector_machine_linear.png', dpi
=300)
```



Solving non-linear problems using a kernel SVM

SVM 可以解决非线性问题

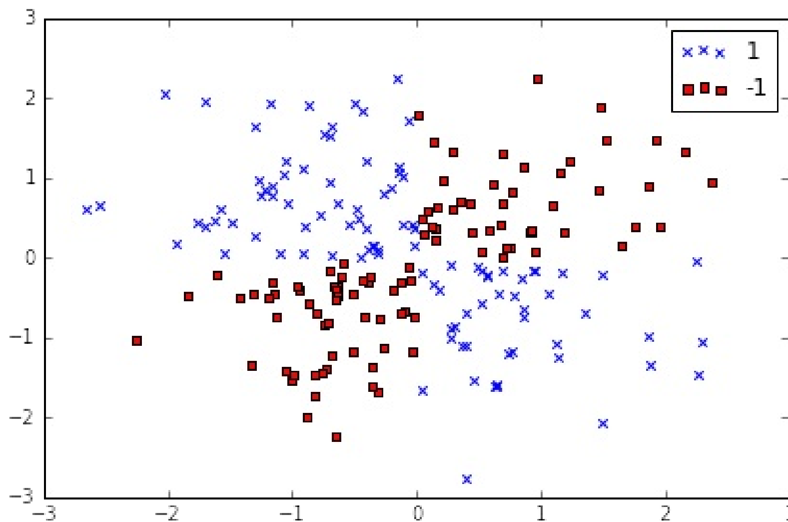
[back to top]

```
# create a simple dataset
np.random.seed(0)
X_xor = np.random.randn(200, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0, X_xor[:, 1] > 0) # 100个
 with label 1, 100 withlabel 0
y_xor = np.where(y_xor, 1, -1)

plt.scatter(X_xor[y_xor==1, 0], X_xor[y_xor==1, 1], c='b', marke
r='x', label='1')
plt.scatter(X_xor[y_xor==-1, 0], X_xor[y_xor==-1, 1], c='r', mar
ker='s', label='-1')

plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('./figures/xor.png', dpi=300)

# 使用普通的  linear logistic Regression 不能很好将样本分为+ve 和-ve
```



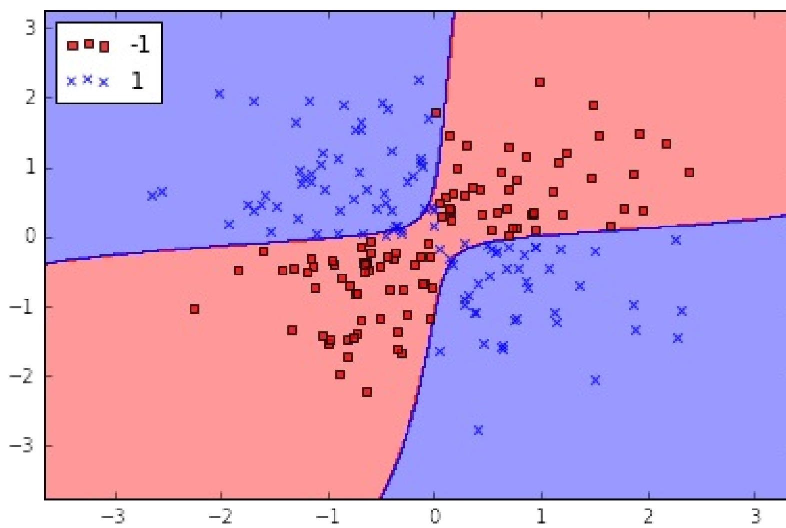
rbf 是指 radial basis function kernel 或者 Gaussian kernel

$$k(x^{(i)}, x^{(j)}) = exp(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2})$$

simplified to $exp(-\gamma\|x^{(i)} - x^{(j)}\|^2)$ with $\gamma = \frac{1}{2\sigma^2}$

```
# 使用 svm kernel 方法, 投射到高纬度中, 使之成为线性可分离的
svm = SVC(kernel='rbf', random_state=0, gamma=0.10, C=10.0)
svm.fit(X_xor, y_xor)
plot_decision_regions(X_xor, y_xor,
                      classifier=svm)

plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/support_vector_machine_rbf_xor.png', dpi=300)
```

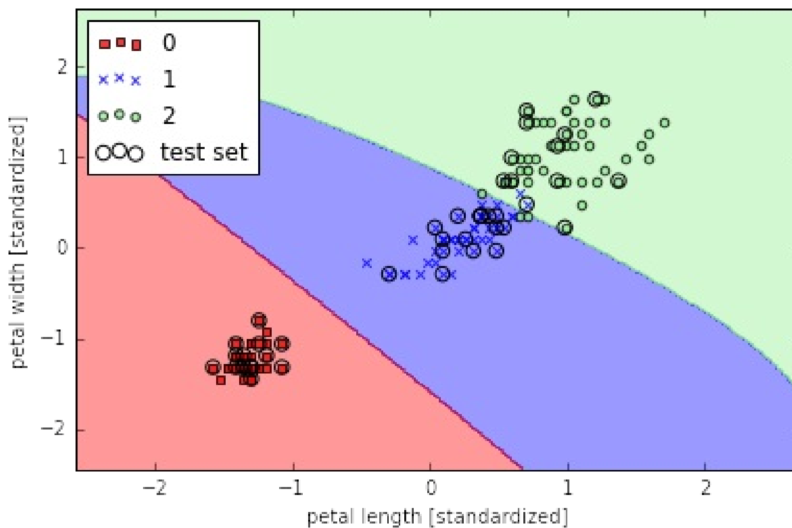


其中γ parameter 可以被理解为 cut-off parameter for Gaussian sphere

当γ增加, 也就增加了训练样本的影响, 也就会使决策边界变得模糊

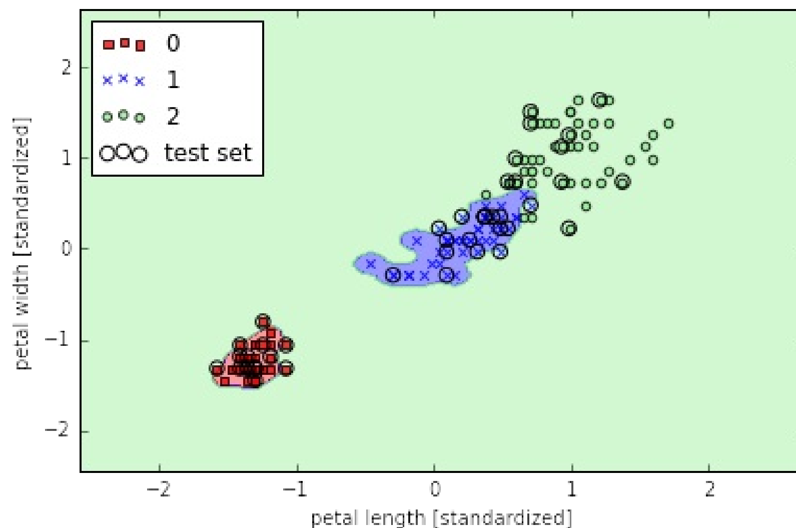
```
from sklearn.svm import SVC
## gamma 较小
svm = SVC(kernel='rbf', random_state=0, gamma=0.2, C=1.0)
svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=svm, test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/support_vector_machine_rbf_iris_1.png',
 dpi=300)
```




```
# gamma 很大, 边界 tight
svm = SVC(kernel='rbf', random_state=0, gamma=100.0, C=1.0)
svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=svm, test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/support_vector_machine_rbf_iris_2.png',
 dpi=300)
```



K-nearest neighbors - a lazy learning algorithm

it doesn't learn a discriminative function from the training data but memorizes the training dataset instead.

1. Choose the number of k and a distance metric.
2. Find the k nearest neighbors of the sample that we want to classify.
3. Assign the class label by majority vote.

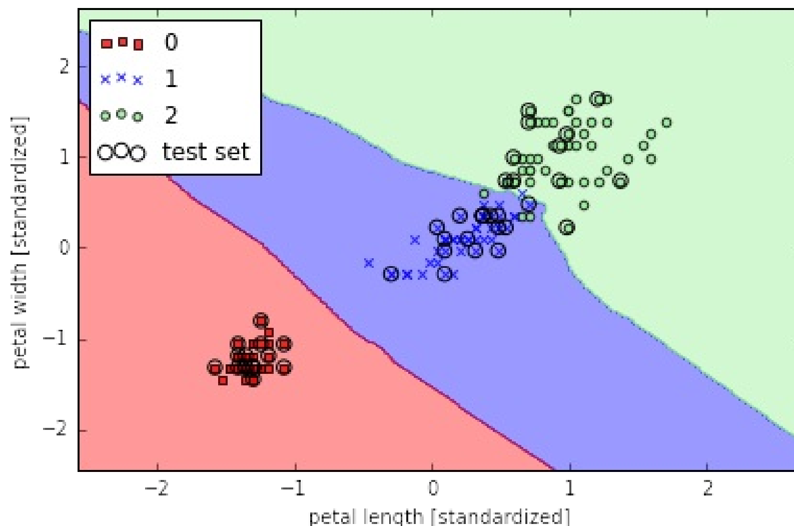
这种方法好处在于新数据进来, 分类器可以马上学习并适应, 但是计算成本也是线性增长, 存储也是问题.

[back to top]

```
from sklearn.neighbors import KNeighborsClassifier
# 寻找5个邻居
knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski'
)
knn.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=knn, test_idx=range(105,150))

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/k_nearest_neighbors.png', dpi=300)
```



如何选择 k 是一个重点, 并且需要标准化数据. 例子中用到的'minkowski' distance 是普通的 Euclidean 和 Manhattan distance 的扩展.

$$d(x^{(i)}, x^{(j)}) = \sqrt[p]{\sum_k \left| x_k^{(i)} x_k^{(j)} \right|^p}$$

Scoring metrics for classification

[back to top]

Classification metrics in Scikit-learn

[back to top]

The `sklearn.metrics` module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values. Most implementations allow each sample to provide a weighted contribution to the overall score, through the `sample_weight` parameter.

Some of these are restricted to the binary classification case:

`matthews_corrcoef` (y_true, y_pred)	Compute the Matthews correlation coefficient (MCC) for binary classes
`precision_recall_curve` (y_true, probas_pred)	Compute precision-recall pairs for different probability thresholds
`roc_curve` (y_true, y_score[, pos_label, ...])	Compute Receiver operating characteristic (ROC)

Others also work in the multiclass case:

`confusion_matrix` (y_true, y_pred[, labels])	Compute confusion matrix to evaluate the accuracy of a classification
`hinge_loss` (y_true, pred_decision[, labels, ...])	Average hinge loss (non-regularized)

Some also work in the multilabel case:

`accuracy_score` (y_true, y_pred[, normalize, ...])	Accuracy classification score.
`classification_report` (y_true, y_pred[, ...])	Build a text report showing the main classification metrics
`f1_score` (y_true, y_pred[, labels, ...])	Compute the F1 score, also known as balanced F-score or F-measure
`fbeta_score` (y_true, y_pred, beta[, labels, ...])	Compute the F-beta score
`hamming_loss` (y_true, y_pred[, classes])	Compute the average Hamming loss.
`jaccard_similarity_score` (y_true, y_pred[, ...])	Jaccard similarity coefficient score
`log_loss` (y_true, y_pred[, eps, normalize, ...])	Log loss, aka logistic loss or cross-entropy loss.
`precision_recall_fscore_support` (y_true, y_pred)	Compute precision, recall, F-measure and support for each class
`precision_score` (y_true, y_pred[, labels, ...])	Compute the precision
`recall_score` (y_true, y_pred[, labels, ...])	Compute the recall
`zero_one_loss` (y_true, y_pred[, normalize, ...])	Zero-one classification loss.

And some work with binary and multilabel (but not multiclass) problems:

`average_precision_score` (y_true, y_score[, ...])	Compute average precision (AP) from prediction scores
`roc_auc_score` (y_true, y_score[, average, ...])	Compute Area Under the Curve (AUC) from prediction scores

```
# 构建数据
from sklearn import datasets
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

X, y = datasets.make_classification(n_classes=2, random_state=0)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train) # standardize by mean & std
X_test_std = sc.transform(X_test)

model = SVC(probability=True, random_state=0)
model.fit(X_train_std, y_train);
```

default score for classification in sklearn is accuracy (标签预测正确的比例)

$accuracy(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} 1(\hat{y}_i = y_i)$ where $1(x)$ is the indicator function

```
model.score(X_test_std, y_test)
```

```
0.83333333333333337
```

```
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test_std)
accuracy_score(y_test, y_pred)
```

```
0.83333333333333337
```

Reading a confusion matrix

[back to top]

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

For multi-class problems, it is often interesting to know which of the classes are hard to predict, and which are easy, or which classes get confused. One way to get more information about misclassifications is `the confusion_matrix`, which shows for each true class, how frequent a given predicted outcome is.

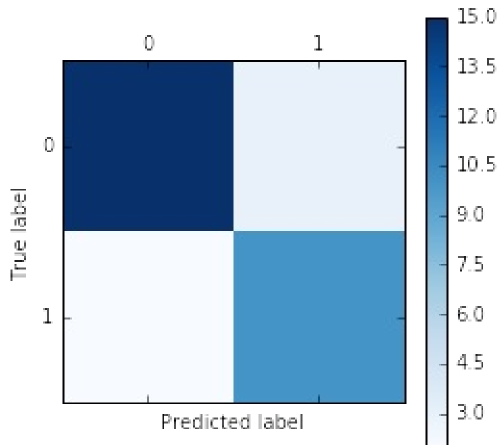
```
from sklearn.metrics import confusion_matrix

y_test_pred = model.predict(X_test_std)

confmat = confusion_matrix(y_test, y_test_pred)
print(confmat)
```

```
[[15  3]
 [ 2 10]]
```

```
plt.matshow(confusion_matrix(y_test, y_test_pred), cmap=plt.cm.B
lues)
plt.colorbar()
plt.xlabel("Predicted label")
plt.ylabel("True label");
```



Precision, recall and F-measures

[back to top]

- **Precision** is how many of the predictions for a class are actually that class.
- **Recall** is how many of the true positives were recovered:
- **f1-score** is the geometric average of precision and recall:

With TP, FP, TN, FN, FPR, TPR standing for "true positive", "false positive", "true negative" and "false negative", "false positive rate", "true positive rate" repectively:

\begin{align} &PRE = \frac{TP}{TP+FP} \ &REC = TPR = \frac{TP}{FN+TP} \ &F1 = 2 \frac{PRE \times REC}{PRE+REC} \ &F_\beta = (1+\beta^2)\frac{PRE \times REC}{\beta^2 PRE+REC} \ &FPR = \frac{FP}{FP+TN} \ &TPR = \frac{TP}{FN+TP} \end{align}

```
from sklearn.metrics import precision_score, recall_score, f1_sc
ore, fbeta_score

print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=
y_test_pred))
print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_test
_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_test_pred))
print('F_beta2: %.3f' % fbeta_score(y_true=y_test, y_pred=y_test
_pred, beta=2))
```

```
Precision: 0.769
Recall: 0.833
F1: 0.800
F_beta2: 0.820
```

Another useful function is the classification_report which provides precision, recall, fscore and support for all classes.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_test_pred))
```

```
             precision    recall  f1-score   support

          0       0.88      0.83      0.86        18
          1       0.77      0.83      0.80        12

avg / total       0.84      0.83      0.83        30
```

These metrics are helpful in two particular cases that come up often in practice:

1. Imbalanced classes, that is one class might be much more frequent than the other.
2. Asymmetric costs, that is one kind of error is much more "costly" than the other.

ROC and AUC

[back to top]

A receiver operating characteristic curve, or ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate.

如果分类器效果很好, 那么图应该会在左上角.

在 ROC curve 的基础上, 可以计算 AUC -- area under the curve.

Area Under Curve

The AUC is a common evaluation metric for binary classification problems. Consider a plot of the true positive rate vs the false positive rate as the threshold value for classifying an item as 0 or is increased from 0 to 1: if the classifier is very good, the true positive rate will increase quickly and the area under the curve will be close to 1. If the classifier is no better than random guessing, the true positive rate will increase linearly with the false positive rate and the area under the curve will be around 0.5.

One characteristic of the AUC is that it is independent of the fraction of the test population which is class 0 or class 1: this makes the AUC useful for evaluating the performance of classifiers on unbalanced data sets.

```
def roc_curve(true_labels, predicted_probs, n_points=100, pos_class=1):
    thr = np.linspace(0,1,n_points)
    tpr = np.zeros(n_points)
    fpr = np.zeros(n_points)

    pos = true_labels == pos_class
    neg = np.logical_not(pos)
    n_pos = np.count_nonzero(pos)
    n_neg = np.count_nonzero(neg)

    for i,t in enumerate(thr):
        tpr[i] = np.count_nonzero(np.logical_and(predicted_probs >= t, pos)) / float(n_pos)
        fpr[i] = np.count_nonzero(np.logical_and(predicted_probs >= t, neg)) / float(n_neg)

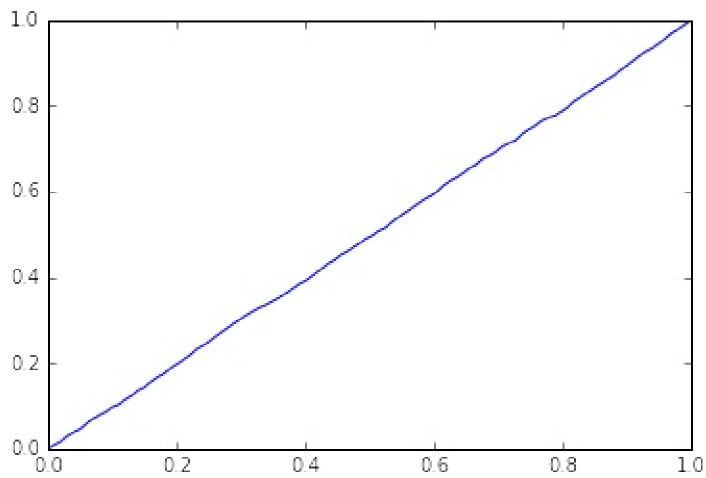
    return fpr, tpr, thr
```

```
df_imputed = pd.read_csv('df_imputed')
```

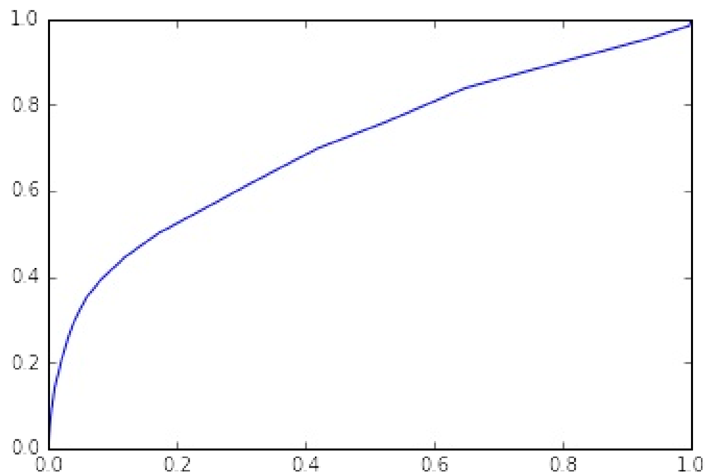
```
features = ['revolving_utilization_of_unsecured_lines',
 'age',
 'number_of_time30-59_days_past_due_not_worse',
 'debt_ratio',
 'monthly_income',
 'number_of_open_credit_lines_and_loans',
 'number_of_times90_days_late',
 'number_real_estate_loans_or_lines',
 'number_of_time60-89_days_past_due_not_worse',
 'number_of_dependents',
 'income_bins',
 'age_bin',
 'monthly_income_scaled']
y = df_imputed.serious_dlqin2yrs
X = pd.get_dummies(df_imputed[features], columns = ['income_bins'
, 'age_bin'])
```

```
from sklearn.cross_validation import train_test_split
train_X, test_X, train_y, test_y = train_test_split(X, y ,train_
size=0.7,random_state=1)
```

```
# Randomly generated predictions should give us a diagonal ROC c
urve
preds = np.random.rand(len(test_y))
fpr, tpr, thr = roc_curve(test_y, preds)
plt.plot(fpr, tpr);
```




```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(train_X,train_y)
preds = clf.predict_proba(test_X)[:,1]
fpr, tpr, thr = roc_curve(test_y, preds)
plt.plot(fpr, tpr);
```



Log loss

[back to top]

Log loss, also called logistic regression loss or cross-entropy loss, is defined on probability estimates. It is commonly used in (multinomial) logistic regression and neural networks, as well as in some variants of expectation-maximization, and can be used to evaluate the probability outputs (`predict_proba`) of a classifier instead of its discrete predictions.

For binary classification with a true label $y \in \{0, 1\}$ and a probability estimate $p = \operatorname{Pr}(y = 1)$, the log loss per sample is the negative log-likelihood of the classifier given the true label:

$$L_{log}(y, p) = -logPr(y|p) = -(ylog(p) + (1 - y)log(1 - p))$$

This extends to the multiclass case as follows. Let the true labels for a set of samples be encoded as a 1-of-K binary indicator matrix Y, i.e., $y_{i,k} = 1$ if sample i has label k taken from a set of K labels. Let P be a matrix of probability estimates, with $p_{i,k} = \operatorname{Pr}(t_{i,k} = 1)$. Then the log loss of the whole set is

$$L_{log}(Y, P) = -logPr(Y|P) = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{k=0}^{K-1} y_{i,k} logp_{i,k}$$

To see how this generalizes the binary log loss given above, note that in the binary case, we have $p_{i,0} = 1 - p_{i,1}$ and $y_{i,0} = 1 - y_{i,1}$, so expanding the inner sum over $y_{i,k} \in \{0, 1\}$ gives the binary log loss.

The log_loss function computes log loss given a list of ground-truth labels and a probability matrix, as returned by an estimator’s predict_proba method.

```
from sklearn.metrics import log_loss
y_true = [0, 0, 1, 1]
y_pred = [[.9, .1], [.8, .2], [.3, .7], [.01, .99]]
log_loss(y_true, y_pred)
```

```
0.1738073366910674‌9
```

Hinge loss

[back to top]

The hinge_loss function computes the average distance between the model and the data using hinge loss, a one-sided metric that considers only prediction errors. (Hinge loss is used in maximal margin classifiers such as support vector machines.)

If the labels are encoded with +1 and -1, y: is the true value, and w is the predicted decisions as output by decision_function, then the hinge loss is defined as:

$$L_{Hinge}(y, w) = max\{1 - wy, 0\} = |1 - wy|_+$$

If there are more than two labels, hinge_loss uses a multiclass variant due to Crammer & Singer. If y_w is the predicted decision for true label and y_t is the maximum of the predicted decisions for all other labels, where predicted decisions are output by decision function, then multiclass hinge loss is defined by:

$$L_{Hinge}(y_w, y_t) = max\{1 + y_t - y_w, 0\}$$

```
# Here a small example demonstrating the use of the hinge_loss f
unction
# with a svm classifier in a binary class problem:
from sklearn import svm
from sklearn.metrics import hinge_loss
X = [[0], [1]]
y = [-1, 1]
est = svm.LinearSVC(random_state=0)
est.fit(X, y)
pred_decision = est.decision_function([[-2], [3], [0.5]])
print(pred_decision)
print(hinge_loss([-1, 1, 1], pred_decision))
```

```
[-2.18173682  2.36360149  0.09093234]
0.303022554204
```

```
# Here is an example demonstrating the use of the hinge_loss fun
ction
# with a svm classifier in a multiclass problem:
X = np.array([[0], [1], [2], [3]])
Y = np.array([0, 1, 2, 3])
labels = np.array([0, 1, 2, 3])
est = svm.LinearSVC()
est.fit(X, Y)
pred_decision = est.decision_function([[-1], [2], [3]])
y_true = [0, 2, 3]
hinge_loss(y_true, pred_decision, labels)
```

```
0.56412359941917456
```

练习：尝试在信贷数据集中使用正则化方法，画出系数的变化，以及最终的预测效果

```

```

Sections

Decision trees learning

[back to top]

Here we'll explore a class of algorithms based on decision trees. Decision trees at their root are extremely intuitive. They encode a series of "if" and "else" choices, similar to how a person might make a decision. However, which questions to ask, and how to proceed for each answer is entirely learned from the data.

For example, if you wanted to create a guide to identifying an animal found in nature, you might ask the following series of questions:

- Is the animal bigger or smaller than a meter long?
  - *bigger*: does the animal have horns?
    - *yes*: are the horns longer than ten centimeters?

        - *no*: is the animal wearing a collar
    - *smaller*: does the animal have two or four legs?
        - *two*: does the animal have wings?
        - *four*: does the animal have a bushy tail?

and so on. This binary splitting of questions is the essence of a decision tree.

One of the main benefit of tree-based models is that they require little preprocessing of the data. They can work with variables of different types (continuous and discrete) and are invariant to scaling of the features.

Another benefit is that tree-based models are what is called "nonparametric", which means they don't have a fix set of parameters to learn. Instead, a tree model can become more and more flexible, if given more data. In other words, the number of free parameters grows with the number of samples and is not fixed, as for example in linear models.

Building a decision tree

[back to top]

```
import numpy as np
from sklearn import datasets
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.3, random_state=0)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train) # standardize by mean & std
X_test_std = sc.transform(X_test)
```

```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
%matplotlib inline

def plot_decision_regions(X, y, classifier, test_idx=None, resol
ution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution)
,
                         np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).
T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

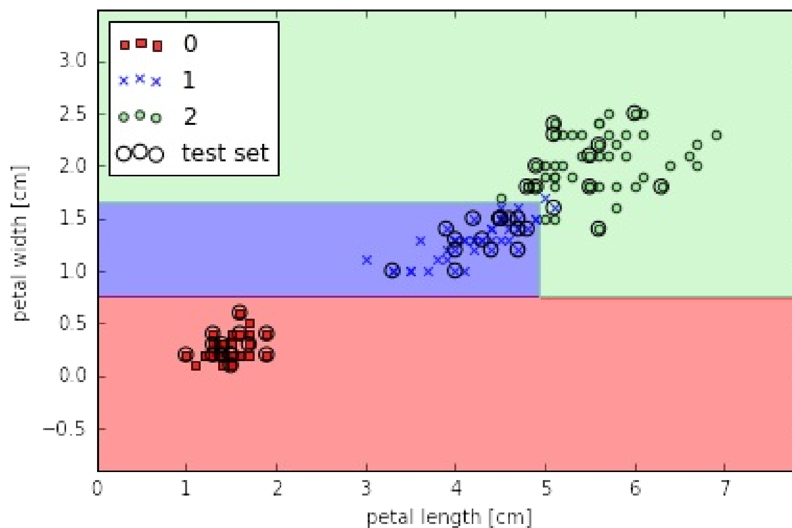
    # plot all samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)

    # highlight test samples
    if test_idx:
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1], c='',
                alpha=1.0, linewidth=1, marker='o',
                s=55, label='test set')
```

```
from sklearn.tree import DecisionTreeClassifier
# max depth 3 using entropy for impurofy
tree = DecisionTreeClassifier(criterion='entropy', max_depth=3,
random_state=0)
tree.fit(X_train, y_train)

X_combined = np.vstack((X_train, X_test))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X_combined, y_combined,
                      classifier=tree, test_idx=range(105,150))

plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/decision_tree_decision.png', dpi=300)
```

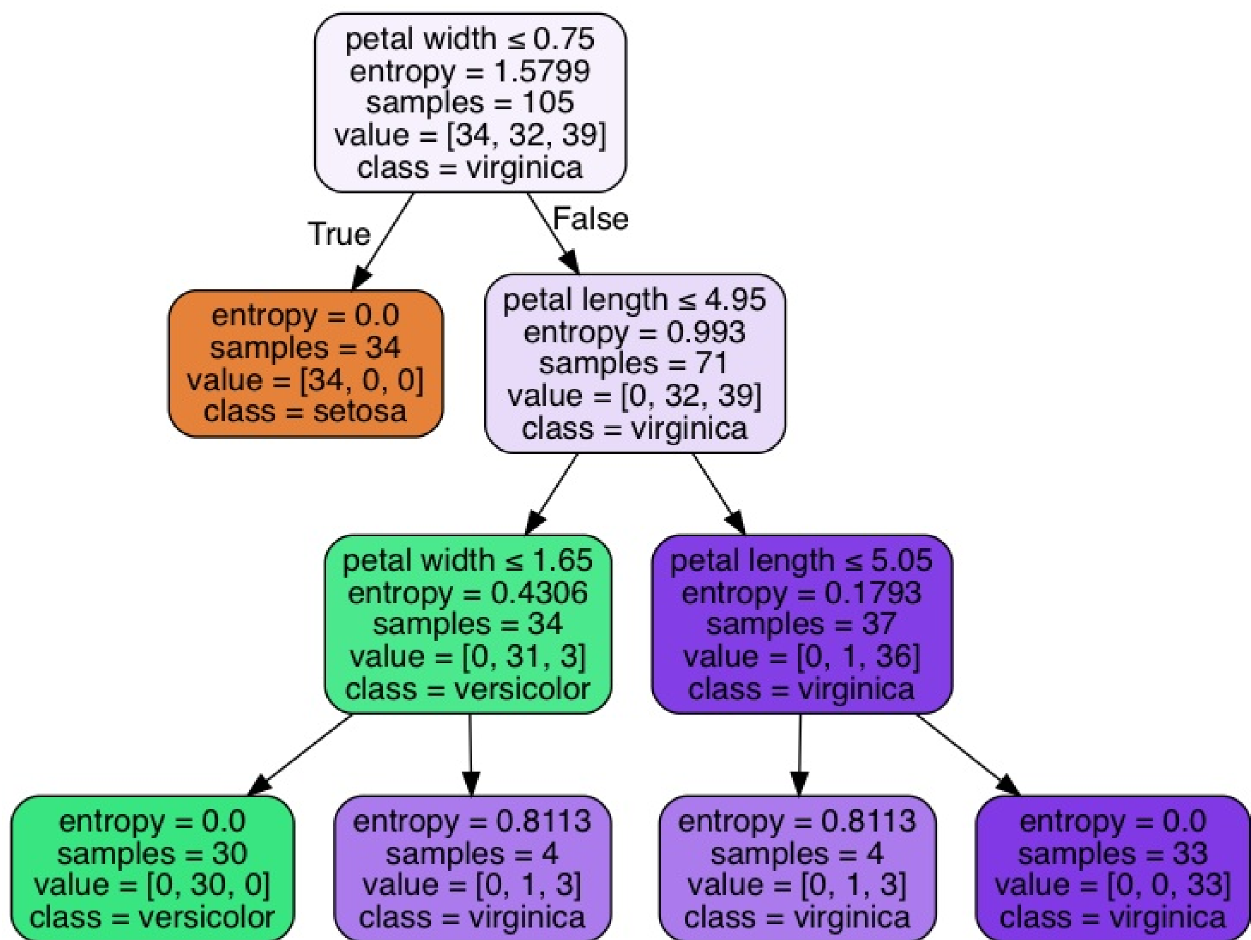


Visualize a decision tree

[back to top]

```
from sklearn.tree import export_graphviz
# export tree as .dot file, install  GraphViz to transfer the fo
rmat
export_graphviz(tree,
                out_file='tree.dot',
                feature_names=['petal length', 'petal width'])
```

```
# pip install pydotplus
import pydotplus
from sklearn.externals.six import StringIO
from IPython.display import Image
dot_data = StringIO()
export_graphviz(tree, out_file=dot_data,
                feature_names=['petal length', 'petal width'],
                class_names=iris.target_names,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



Different impurity criteria

[back to top]

```
def gini(p):
    return (p)*(1 - (p)) + (1-p)*(1 - (1-p))

def entropy(p):
    return - p*np.log2(p) - (1 - p)*np.log2((1 - p))

def error(p):
    return 1 - np.max([p, 1 - p])

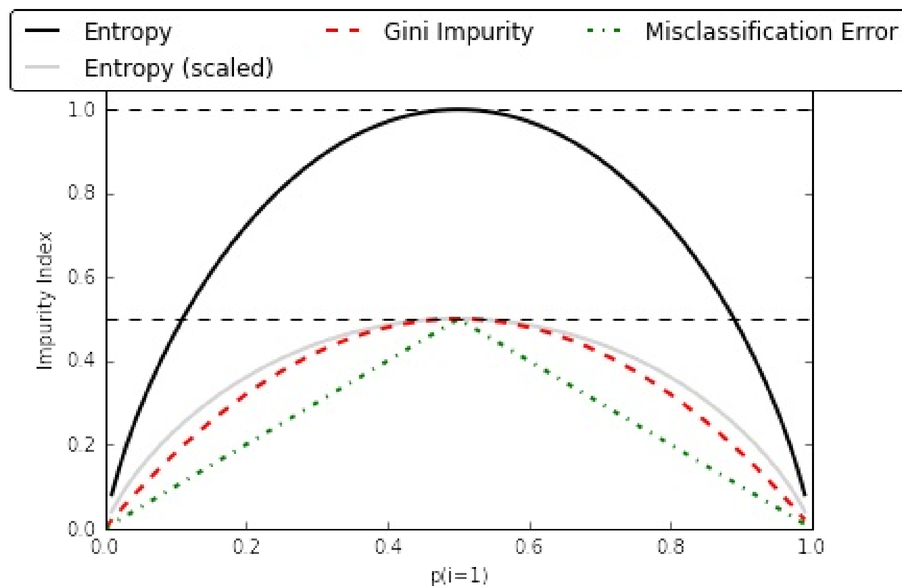
x = np.arange(0.0, 1.0, 0.01)

ent = [entropy(p) if p != 0 else None for p in x]
sc_ent = [e*0.5 if e else None for e in ent]
err = [error(i) for i in x]


fig = plt.figure()
ax = plt.subplot(111)
for i, lab, ls, c, in zip([ent, sc_ent, gini(x), err],
                          ['Entropy', 'Entropy (scaled)',
                           'Gini Impurity', 'Misclassification Error'],
                          ['-', '-', '--', '-.'],
                          ['black', 'lightgray', 'red', 'green', 'cyan']
):
    line = ax.plot(x, i, label=lab, linestyle=ls, lw=2, color=c)


# 画图
ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.15),
          ncol=3, fancybox=True, shadow=False)

ax.axhline(y=0.5, linewidth=1, color='k', linestyle='--')
ax.axhline(y=1.0, linewidth=1, color='k', linestyle='--')
plt.ylim([0, 1.1])
plt.xlabel('p(i=1)')
plt.ylabel('Impurity Index')
plt.tight_layout()
# plt.savefig('./figures/impurity.png', dpi=300, bbox_inches='ti
ght')
```



Implement a binary decision tree in python

[back to top]

```
from collections import Counter
import numpy as np


# 构建一个类，来表征二分树的结构
# Tree 里的属性除了包括左右节点的 Tree 之外，还有此节点中包括数据的标签及
其熵值，然后还有要切分 feature 的 idex
class Tree:
    """ Binary Tree"""
    def __init__(self, labels, split_idx=None,
                 children_left=None, children_right=None):
        self.children_left = children_left
        self.children_right = children_right
        self.labels = labels
        self.split_idx = split_idx
        self.entropy = calc_entropy(self.labels)

    def predict(self):
        most_freq = np.bincount(self.labels).argmax()  # find mo
```

```
st frequent element
        return most_freq

    def __repr__(self, level=0):
        """ make it easy to visualize a tree"""
        prefix = "\t" * level
        string = prefix + "entropy = {}, labels = {}, [0s, 1s] =
 {}\n".format(
            self.entropy, self.labels, np.bincount(self.labels,
minlength=2))
        if self.split_idx is not None:
            string += prefix + "split on Column {}\n".format(sel
f.split_idx)
            string += prefix + "True:\n"
            string += self.children_left.__repr__(level+1)
            string += prefix + "False:\n"
            string += self.children_right.__repr__(level+1)
        return string


# 计算一组数据里的熵值
def calc_entropy(labels):
    """ calculate entropy from an array of labels"""
    size = float(len(labels))
    cnt = Counter(labels)
    entropy = 0
    for label in set(labels):
        prob = cnt[label] / size
        entropy += -1 * prob * np.log2(prob)
    return entropy


# 不同的决策树算法（如 ID3, C4.5, CART 等）会用不同的标准来选择要切分的
feature
# 这里使用的是 Information Gain，即 feature 切分前后的熵值变化
def choose_best_feature_to_split(features, labels):
    """ choose the best split feature which maximize information
 gain """
    num_features = features.shape[1]
    base_entropy = calc_entropy(labels)
```

```
    best_info_gain = 0
    best_feature = None

    for i in range(num_features):
        new_entropy = 0
        for value in [0, 1]:
            new_labels = labels[features[:, i] == value]
            weight = float(len(new_labels)) / len(labels)
            new_entropy += weight * calc_entropy(new_labels)
        info_gain = base_entropy - new_entropy
        if info_gain > best_info_gain:
            best_info_gain = info_gain
            best_feature = i
    return best_feature


def create_decision_tree(features, labels,
                         current_depth=0, max_depth=10):
    """ recursively create tree """
    tree = Tree(labels)

    # define stop condition
    # stop when all data in this node are from the same class
    if len(set(labels)) == 1:
        return tree
    # stop when max_depth are reached
    if current_depth >= max_depth:
        return tree

    # split on the best feature found
    best_feature = choose_best_feature_to_split(features, labels
)
    if best_feature is None:
        return tree

    # recursively build subtrees
    msk = (features[:, best_feature] == 1)
    tree.split_idx = best_feature
    tree.children_left = create_decision_tree(
```

```
        features[msk], labels[msk], current_depth+1)
    tree.children_right = create_decision_tree(
        features[~msk], labels[~msk], current_depth+1)
    return tree
```

```
# 模拟一组数据来测试
data = np.array([[1, 0],
                 [1, 1],
                 [0, 1],
                 [0, 0]])
labels = np.array([1, 0, 0, 0])


tree = create_decision_tree(data, labels)
```

```
tree
```

```
entropy = 0.811278124459, labels = [1 0 0 0], [0s, 1s] = [3 1]
split on Column 0
True:
    entropy = 1.0, labels = [1 0], [0s, 1s] = [1 1]
    split on Column 1
    True:
        entropy = 0.0, labels = [0], [0s, 1s] = [1 0]
    False:
        entropy = 0.0, labels = [1], [0s, 1s] = [0 1]
False:
    entropy = 0.0, labels = [0 0], [0s, 1s] = [2 0]
```

```
# 遍历二分树，来得到分类预测
def _classify(tree, data_row):
    """ prediction for new data"""
    if tree.split_idx is None: # if it's a leaf
        return tree.predict()

    split_idx = tree.split_idx
    if data_row[split_idx]:  # if True for split condition
        return _classify(tree.children_left, data_row)
    else:
        return _classify(tree.children_right, data_row)

def classify(tree, data):
    data = np.array(data)
    num_row = data.shape[0]
    results = np.empty(shape=num_row)
    for i in range(num_row):
        results[i] = _classify(tree, data[i, :])
    return results
```

```
new_data = [[0, 0],
            [1, 0]]

classify(tree, new_data)
```

```
array([ 0.,  1.])
```

Combining weak to strong learners via random forests

[back to top]

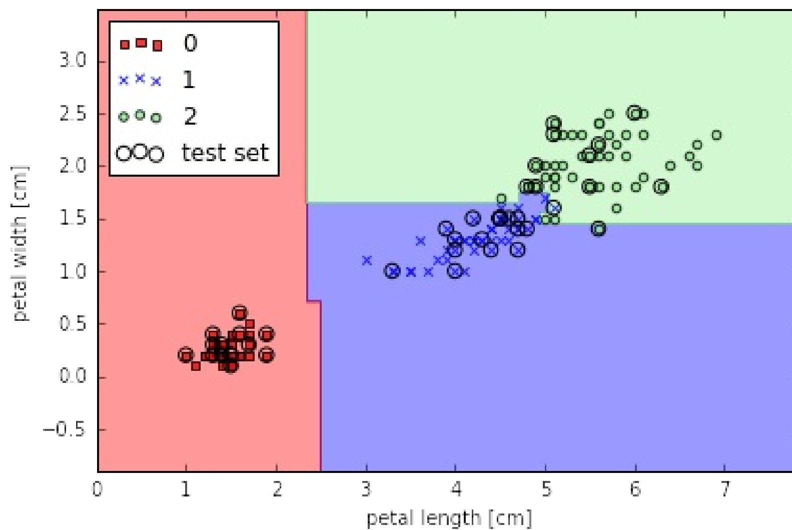
可以看做是 ensemble of decision trees, 将弱的模型结合在一起变成强模型. 更易扩展, 且较少会 overfitting.

1. draw a random **bootstrap** sample of size n (with replacement)
2. grow decision tree from bootstrap sample, at each node:
    - randomly select d features without replacement
    - split node using feature that provides best split
3. repeat 1 & 2 k times.
4. aggregate the prediction by each tree to assign the class label by **majority vote**

```
from sklearn.ensemble import RandomForestClassifier
# from 10 decision trees, n_jobs 值使用 cpu 个数
forest = RandomForestClassifier(criterion='entropy',
                                n_estimators=10,
                                random_state=1,
                                n_jobs=2)
forest.fit(X_train, y_train)

plot_decision_regions(X_combined, y_combined,
                      classifier=forest, test_idx=range(105,150)
)

plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/random_forest.png', dpi=300)
```



Learning with ensembles

将一系列分类器集合起来, 取多数为分类结果

Build powerful models from weak learners that learn from their mistakes

ensemble method 就是讲多个不同的分类器集合组合为一个大的分类器. 选择最终结果是以 majority voting

即使每个单独的分类器错误率较高, 但将多个分类器组合之后, 错误率就会大大降低

[back to top]

假设我们组合了 n 个分类器，它们的错误率都为 ε, 各个分类器之间独立。
则这 n 个分类器里, 多于 k 个分类器分类错误的概率为
$P(y \geq k) = \sum_k^n \binom{n}{k} \varepsilon^k (1-\varepsilon)^{n-k}$

```
from scipy.misc import comb
import math

# emsemble error rate
def ensemble_error(n_classifier, error):
    k_start = int(math.ceil(n_classifier / 2.0))
    probs = [comb(n_classifier, k) * error**k *
                 (1-error)**(n_classifier - k)
                for k in range(k_start, n_classifier + 1)]
    return sum(probs)
```

```
# 11个分类器，每个分类器的 error rate 是0.25的话，通过 combinator 之后的 error rate
ensemble_error(n_classifier=11, error=0.25)
```

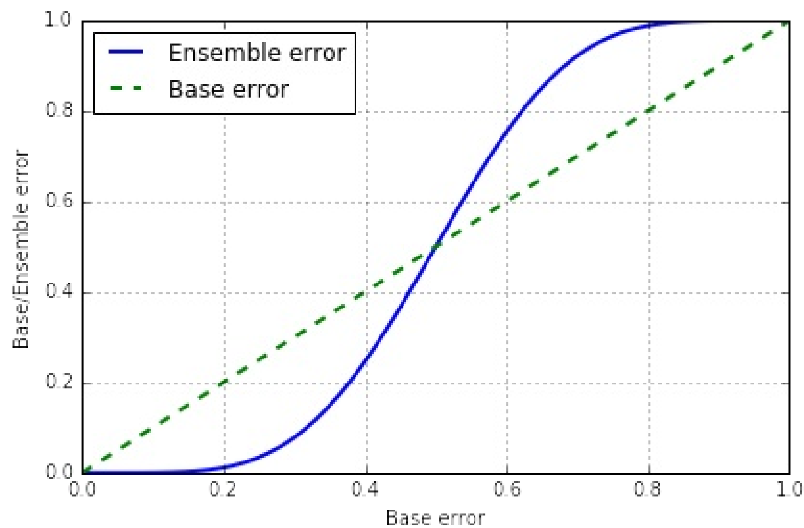
```
0.034327507019042969
```

```
# ensemble error 和 base error 的关系
error_range = np.arange(0.0, 1.01, 0.01)
ens_errors = [ensemble_error(n_classifier=11, error=error)
              for error in error_range]

plt.plot(error_range, ens_errors,
         label='Ensemble error', linewidth=2)

plt.plot(error_range, error_range,
         linestyle='--', label='Base error',linewidth=2)

plt.xlabel('Base error')
plt.ylabel('Base/Ensemble error')
plt.legend(loc='upper left')
plt.grid()
plt.tight_layout()
# plt.savefig('./figures/ensemble_err.png', dpi=300)
```



ε < 0.5 时, emsemble error 都要低于 base error, ε > 0.5 时, emsemble error 就会大于 base error

Majority vote classifier

combine different classication algorithms associated with individual weights for confidence

[back to top]

当多个分类器 C 拥有相同权重时，ensemble 给出的预测 $\hat{y}$ 为众数：
$\hat{y} = mode\{C_1(x), C_2(x), \ldots, C_m(x)\}$

若分类器 C_j 对应不同的权重 w_j, 则 $\hat{y} = \arg\max_i \sum_{j=1}^{m} w_j P_{ij}$ 其中 P_{ij} 是 C_j 预测结果为 i 的概率

```
import numpy as np
np.argmax(np.bincount([0, 0, 1],
                      weights=[0.2, 0.2, 0.6]))

# np.argmax: returns the indices of the maximum values along an
axis.
# np.bincount: Count number of occurrences of each value in arra
y of non-negative ints
```

```
1
```

```
np.bincount([0, 0, 1],
                      weights=[0.2, 0.2, 0.6])
```

```
array([ 0.4,  0.6])
```

```
ex = np.array([[0.9, 0.1],  # C1 的预测结果
               [0.8, 0.2],  # C2 的预测结果
               [0.4, 0.6]]) # C3 的预测结果

p = np.average(ex,
               axis=0,
               weights=[0.2, 0.2, 0.6])  # C1, C2, C3 的权重
p
```

```
array([ 0.58,  0.42])
```

$p(i_0|x) = 0.58$ $p(i_1|x) = 0.42$ $\hat{y} = \arg\max_i [p(i_0|x), p(i_1|x)] = 0$

```
np.argmax(p)
```

```
0
```

VotingClassifier in Sklearn

使用 Sklearn 中自带的 VotingClassifier

[back to top]

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier

# 使用 3 个分类器
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()

# 生成数据
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2
]])
y = np.array([1, 1, 1, 2, 2, 2])

# voting='hard', uses predicted class labels for majority rule v
oting.
eclf1 = VotingClassifier(estimators=[
        ('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting='hard'
)
eclf1 = eclf1.fit(X, y)
print(eclf1.predict(X))

# voting='soft', predicts the class label based on the argmax of
 the sums of the predicted probalities
eclf2 = VotingClassifier(estimators=[
        ('lr', clf1), ('rf', clf2), ('gnb', clf3)],
        voting='soft')
eclf2 = eclf2.fit(X, y)
print(eclf2.predict(X))

# add weight
eclf3 = VotingClassifier(estimators=[
       ('lr', clf1), ('rf', clf2), ('gnb', clf3)],
       voting='soft', weights=[2,1,1])
eclf3 = eclf3.fit(X, y)
print(eclf3.predict(X))
```

```
[1 1 1 2 2 2]
[1 1 1 2 2 2]
[1 1 1 2 2 2]
```

Combining different algorithms for classification with majority vote

[back to top]

```
from sklearn import datasets
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

# load iris data
iris = datasets.load_iris()
X, y = iris.data[50:, [1, 2]], iris.target[50:]

st = StandardScaler()
X = st.fit_transform(X)

le = LabelEncoder()
y = le.fit_transform(y)

# 50% train, 50% test
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.5, random_state=1)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import cross_val_score

clf1 = LogisticRegression(C=0.01, random_state=42)
clf2 = KNeighborsClassifier(n_neighbors=1)
clf3 = DecisionTreeClassifier(max_depth=1, random_state=42)

clf_labels = ['Logistic Regression', 'KNN', 'Decision Tree']
all_clf = [clf1, clf2, clf3]

print('10-fold cross validation:\n')
for clf, label in zip(all_clf, clf_labels):
    scores = cross_val_score(estimator=clf, X=X_train, y=y_train
,
                             cv=10, scoring='roc_auc')
    print("ROC AUC: %0.2f (+/- %0.2f) [%s]"
             % (scores.mean(), scores.std(), label))
```

```
10-fold cross validation:

ROC AUC: 0.93 (+/- 0.15) [Logistic Regression]
ROC AUC: 0.93 (+/- 0.10) [KNN]
ROC AUC: 0.92 (+/- 0.15) [Decision Tree]
```

```
from sklearn.ensemble import VotingClassifier

mv_clf = VotingClassifier(
    estimators=[('c1', clf1), ('c2', clf2), ('c3', clf3)], votin
g='soft')

clf_labels += ['Majority Voting']
all_clf += [mv_clf]

print('10-fold cross validation:\n')
for clf, label in zip(all_clf, clf_labels):
    scores = cross_val_score(estimator=clf, X=X_train, y=y_train
,
                             cv=10, scoring='roc_auc')
    print("ROC AUC: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
10-fold cross validation:

ROC AUC: 0.93 (+/- 0.15) [Logistic Regression]
ROC AUC: 0.93 (+/- 0.10) [KNN]
ROC AUC: 0.92 (+/- 0.15) [Decision Tree]
ROC AUC: 0.97 (+/- 0.10) [Majority Voting]
```

最后一个是 majority voting, 明显比单独的分类器结果好

Evaluating the ensemble classifier

[back to top]

在测试集上评估各个分类器的 ROC AUC

```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

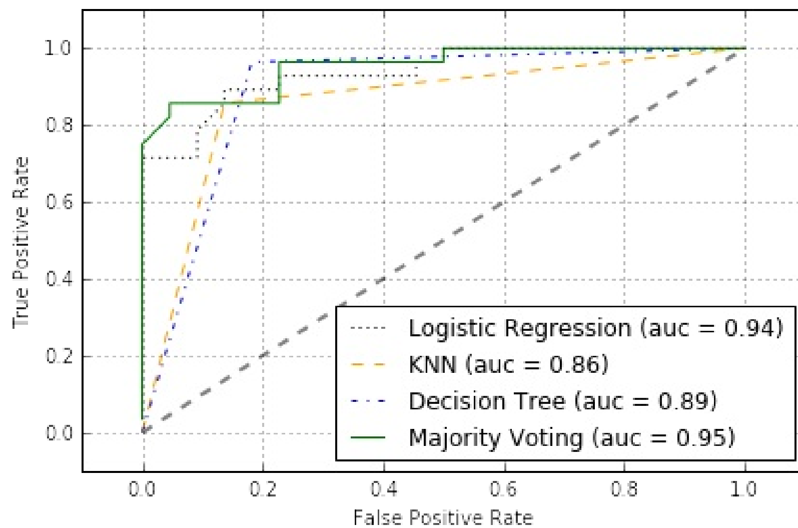
colors = ['black', 'orange', 'blue', 'green']
linestyles = [':', '--', '-.', '-']
for clf, label, clr, ls \
        in zip(all_clf, clf_labels, colors, linestyles):

    # assuming the label of the positive class is 1
    y_pred = clf.fit(X_train, y_train).predict_proba(X_test)[:, 1
]
    fpr, tpr, thresholds = roc_curve(y_true=y_test,
                                     y_score=y_pred)
    roc_auc = auc(x=fpr, y=tpr)
    plt.plot(fpr, tpr,
             color=clr,
             linestyle=ls,
             label='%s (auc = %0.2f)' % (label, roc_auc))

plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1],
         linestyle='--',
         color='gray',
         linewidth=2)

plt.xlim([-0.1, 1.1])
plt.ylim([-0.1, 1.1])
plt.grid()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.tight_layout()
# plt.savefig('./figures/roc.png', dpi=300)
```



ROC 可看出, ensemble classfifier 在 test set 上表现不错

对比决策边界

```
from itertools import product

x_min = X_train[:, 0].min() - 1
x_max = X_train[:, 0].max() + 1
y_min = X_train[:, 1].min() - 1
y_max = X_train[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# 添加 subplots
f, axarr = plt.subplots(nrows=2, ncols=2,
                        sharex='col', sharey='row',
                        figsize=(7, 5))

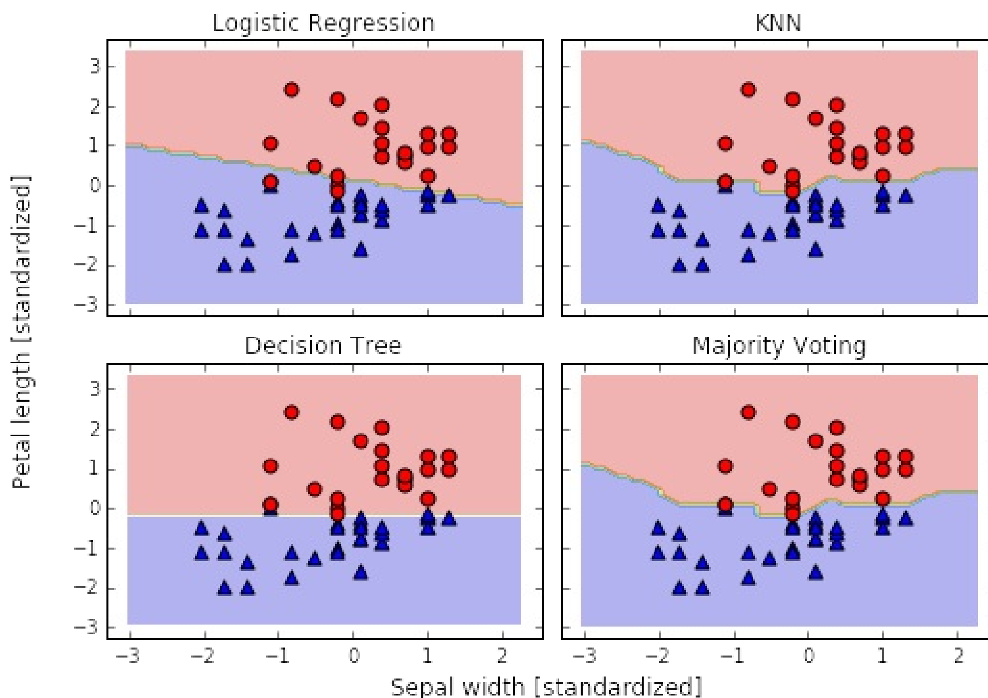
for idx, clf, tt in zip(product([0, 1], [0, 1]),
                        all_clf, clf_labels):
    clf.fit(X_train, y_train)
    Z = clf.predict(np.vstack([xx.ravel(), yy.ravel()]).T)
    Z = Z.reshape(xx.shape)

    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.3)
    axarr[idx[0], idx[1]].scatter(X_train[y_train==0, 0],
                                  X_train[y_train==0, 1],
```

```
                              c='blue', marker='^', s=50)
    axarr[idx[0], idx[1]].scatter(X_train[y_train==1, 0],
                              X_train[y_train==1, 1],
                              c='red', marker='o', s=50)
    axarr[idx[0], idx[1]].set_title(tt)

plt.text(-3.5, -4.5,
         s='Sepal width [standardized]',
         ha='center', va='center', fontsize=12)
plt.text(-10.5, 4.5,
         s='Petal length [standardized]',
         ha='center', va='center',
         fontsize=12, rotation=90)

plt.tight_layout()
# plt.savefig('./figures/voting_panel', bbox_inches='tight', dpi
=300)
```



Bagging -- Building an ensemble of

classifiers from bootstrap samples

- draw `bootstrap` samples (random samples with replacement) from initial training set
- random forests are a special case of bagging where we also use random feature subsets to fit the individual decision trees

[back to top]

```
import pandas as pd
# wine dataset
df_wine = pd.read_csv('ftp://ftp.ics.uci.edu/pub/machine-learnin
g-databases/wine/wine.data',
                      header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash'
,
'Alcalinity of ash', 'Magnesium', 'Total phenols',
'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proli
ne']

# only consider Wine classes 2 and 3
df_wine = df_wine[df_wine['Class label'] != 1]

y = df_wine['Class label'].values
X = df_wine[['Alcohol', 'Hue']].values
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.cross_validation import train_test_split

# 转换 label
le = LabelEncoder()
y = le.fit_transform(y)

# 60% train, 40% test
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.40, random_state=1)
```

```
# sklearn 提供的 BaggingClassifier, 其实功能已经超过 Bagging 了
# 它既能对 samples 采样，也能对 features 采样
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(criterion='entropy')

# 用 Decision Tree 作 base
bag = BaggingClassifier(base_estimator=tree,
                        n_estimators=500,
                        max_samples=1.0,   # 子采样 samples 的比例
                        max_features=1.0,  # 子采样 features 的比例

                        bootstrap=True,    # 采样 samples 时是否使
用 bootstrap
                        bootstrap_features=False,  # 采样 feature
s 时是否使用 bootstrap
                        random_state=1)
```

```
from sklearn.metrics import accuracy_score

tree = tree.fit(X_train, y_train)
y_train_pred = tree.predict(X_train)
y_test_pred = tree.predict(X_test)

tree_train = accuracy_score(y_train, y_train_pred)
tree_test = accuracy_score(y_test, y_test_pred)
print('Decision tree train/test accuracies %.3f/%.3f'
      % (tree_train, tree_test))

bag = bag.fit(X_train, y_train)
y_train_pred = bag.predict(X_train)
y_test_pred = bag.predict(X_test)

bag_train = accuracy_score(y_train, y_train_pred)
bag_test = accuracy_score(y_test, y_test_pred)
print('Bagging train/test accuracies %.3f/%.3f'
      % (bag_train, bag_test))
```

```
Decision tree train/test accuracies 1.000/0.854
Bagging train/test accuracies 1.000/0.896
```

使用 Bagging 之后，测试集的准确率有提升

```
# 画决策边界
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x_min = X_train[:, 0].min() - 1
x_max = X_train[:, 0].max() + 1
y_min = X_train[:, 1].min() - 1
y_max = X_train[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
```

```
f, axarr = plt.subplots(nrows=1, ncols=2,
                        sharex='col',
                        sharey='row',
                        figsize=(8, 3))


for idx, clf, tt in zip([0, 1],
                        [tree, bag],
                        ['Decision Tree', 'Bagging']):
    clf.fit(X_train, y_train)

    Z = clf.predict(np.vstack([xx.ravel(), yy.ravel()]).T)
    Z = Z.reshape(xx.shape)

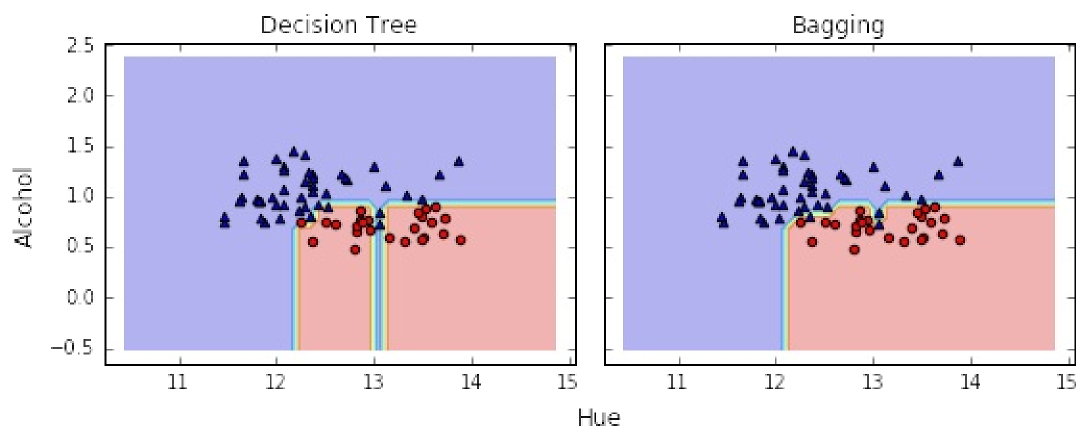
    axarr[idx].contourf(xx, yy, Z, alpha=0.3)
    axarr[idx].scatter(X_train[y_train==0, 0],
                       X_train[y_train==0, 1],
                       c='blue', marker='^')

    axarr[idx].scatter(X_train[y_train==1, 0],
                       X_train[y_train==1, 1],
                       c='red', marker='o')

    axarr[idx].set_title(tt)

axarr[0].set_ylabel('Alcohol', fontsize=12)
plt.text(10.2, -1.2, s='Hue',
         ha='center', va='center', fontsize=12)

plt.tight_layout()
#plt.savefig('./figures/bagging_region.png',
#            dpi=300,
#            bbox_inches='tight')
```



Bagging 减少了 overfitting， 使决策边界更平滑

Leveraging of weak learners via adaptive boosting

let the weak learners subsequently learn from misclassified training samples to improve the performance of the ensemble

[back to top]

Adaptive boosting (AdaBoost)

1. Set weight vector w to uniform weights where $\sum_i w_i = 1$
2. For j in m boosting rounds, do the following:
3. Train a weighted weak learner: $C_j = train(X, y, w)$.
4. Predict class labels: $\hat{y} = predict(C_j, X)$.
5. Compute weighted error rate: $\epsilon = w \cdot (\hat{y} \neq y)$.
6. Compute coefficient: $\alpha_j = \frac{1}{2} ln \frac{1-\epsilon}{\epsilon}$.
7. Update weights: $w := w \times exp(-\alpha_j \times \hat{y} \times y)$.
8. Normalize weights to sum to 1: $w := \frac{w}{\sum_i w_i}$.
9. Compute final prediction: $\hat{y} = (\sum_{j=1}^{m} (\alpha_j \times predict(C_j, X)) > 0)$.

· denotes dot product between two vectors
× denotes element-wise multiplication of two vectors

```
from sklearn.ensemble import AdaBoostClassifier

tree = DecisionTreeClassifier(criterion='entropy',
                              max_depth=1)

ada = AdaBoostClassifier(base_estimator=tree,
                         n_estimators=500,
                         learning_rate=0.1,
                         random_state=0)
```

```
tree = tree.fit(X_train, y_train)
y_train_pred = tree.predict(X_train)
y_test_pred = tree.predict(X_test)

tree_train = accuracy_score(y_train, y_train_pred)
tree_test = accuracy_score(y_test, y_test_pred)
print('Decision tree train/test accuracies %.3f/%.3f'
      % (tree_train, tree_test))

ada = ada.fit(X_train, y_train)
y_train_pred = ada.predict(X_train)
y_test_pred = ada.predict(X_test)

ada_train = accuracy_score(y_train, y_train_pred)
ada_test = accuracy_score(y_test, y_test_pred)
print('AdaBoost train/test accuracies %.3f/%.3f'
      % (ada_train, ada_test))
```

```
Decision tree train/test accuracies 0.845/0.854
AdaBoost train/test accuracies 1.000/0.875
```

Adaboost 可以减少 Bias，但可能引入更多的 Variance

```
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(1, 2, sharex='col', sharey='row', figsiz
e=(8, 3))

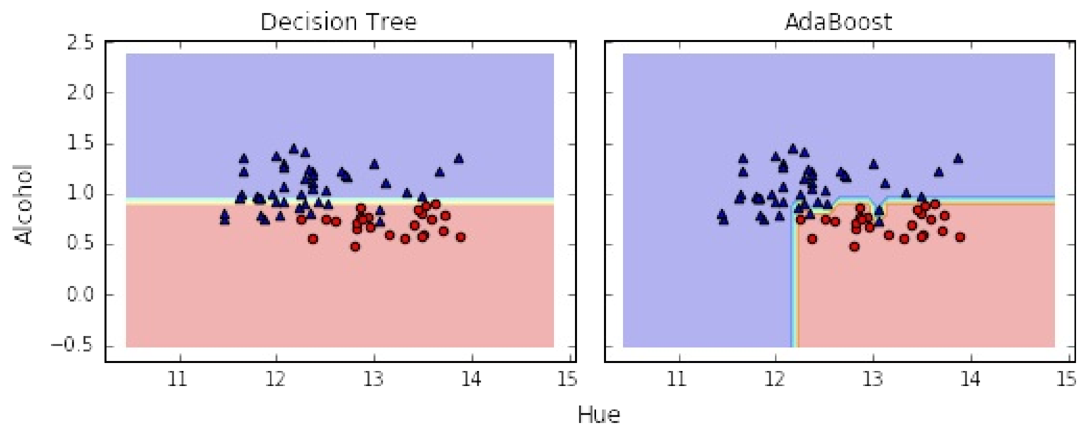

for idx, clf, tt in zip([0, 1],
                        [tree, ada],
                        ['Decision Tree', 'AdaBoost']):
    clf.fit(X_train, y_train)

    Z = clf.predict(np.vstack([xx.ravel(), yy.ravel()]).T)
    Z = Z.reshape(xx.shape)

    axarr[idx].contourf(xx, yy, Z, alpha=0.3)
    axarr[idx].scatter(X_train[y_train==0, 0],
                       X_train[y_train==0, 1],
                       c='blue', marker='^')
    axarr[idx].scatter(X_train[y_train==1, 0],
                       X_train[y_train==1, 1],
                       c='red', marker='o')
    axarr[idx].set_title(tt)

axarr[0].set_ylabel('Alcohol', fontsize=12)
plt.text(10.2, -1.2, s='Hue',
         ha='center', va='center', fontsize=12)

plt.tight_layout()
#plt.savefig('./figures/adaboost_region.png',
#            dpi=300,
#            bbox_inches='tight')
```



Adaboost 的决策边界比 tree 复杂, 与 BaggingClassifier 相似.

Ensemble method 需要更多的计算资源, 这个在实际运用中也是要考虑的.

Algorithm implementation

[back to top]

广义提升树算法详解

```
import numpy
import matplotlib.pyplot as plot
%matplotlib inline
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from math import floor
import random
```

```
# Build a simple data set with y = x + random
n_points = 1000

# x values for plotting
x_plot = [(float(i) / float(n_points) - 0.5) for i in range(n_po
ints + 1)]
```

```
# x needs to be list of lists.
x = [[s] for s in x_plot]

# y (labels) has random noise added to x-value
# set seed
numpy.random.seed(1)
y = [s + numpy.random.normal(scale=0.1) for s in x_plot]

# take fixed test set 30% of sample
n_sample = int(n_points * 0.30)
idx_test = random.sample(range(n_points), n_sample)
idx_test.sort()
idx_train = [idx for idx in range(n_points) if not (idx in idx_t
est)]

# Define test and training attribute and label sets
x_train = [x[r] for r in idx_train]
x_test = [x[r] for r in idx_test]
y_train = [y[r] for r in idx_train]
y_test = [y[r] for r in idx_test]

# train a series of models on random subsets of the training data

# collect the models in a list and check error of composite as l
ist grows

# maximum number of models to generate
num_trees_max = 30

# tree depth - typically at the high end
tree_depth = 5

# initialize a list to hold models
mode_list = []
pred_list = []
eps = 0.3

# initialize residuals to be the labels y
residuals = list(y_train)
```

```
for i_trees in range(num_trees_max):
    mode_list.append(DecisionTreeRegressor(max_depth=tree_depth)
)
    mode_list[-1].fit(x_train, residuals)

    # make prediction with latest model and add to list of predi
ctions
    latest_in_sample_prediction = mode_list[-1].predict(x_train)

    # use new predictions to update residuals
    residuals = [residuals[i] - eps * latest_in_sample_predictio
n[i]
                 for i in range(len(residuals))]

    latest_out_sample_prediction = mode_list[-1].predict(x_test)
    pred_list.append(list(latest_out_sample_prediction))

# build cumulative prediction from first "n" models
mse = []
all_predictions = []
for i_models in range(len(mode_list)):

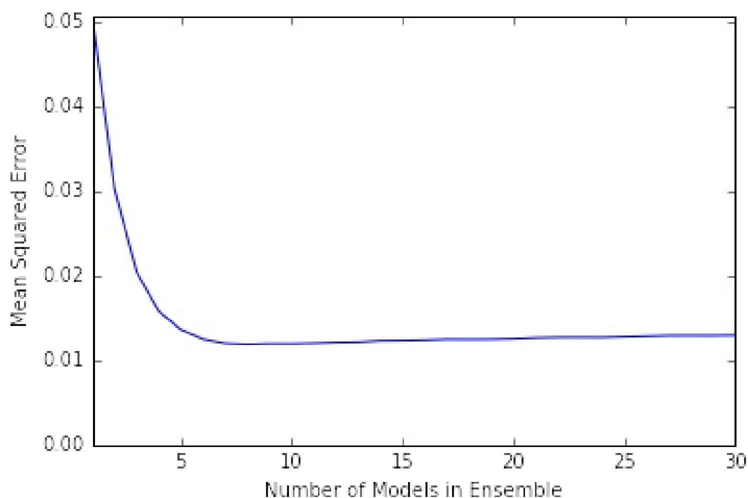
    # add the first "i_models" of the predictions and multiply b
y eps
    prediction = []
    for i_pred in range(len(x_test)):
        prediction.append(
            sum([pred_list[i][i_pred] for i in range(i_models + 1
)]) * eps)

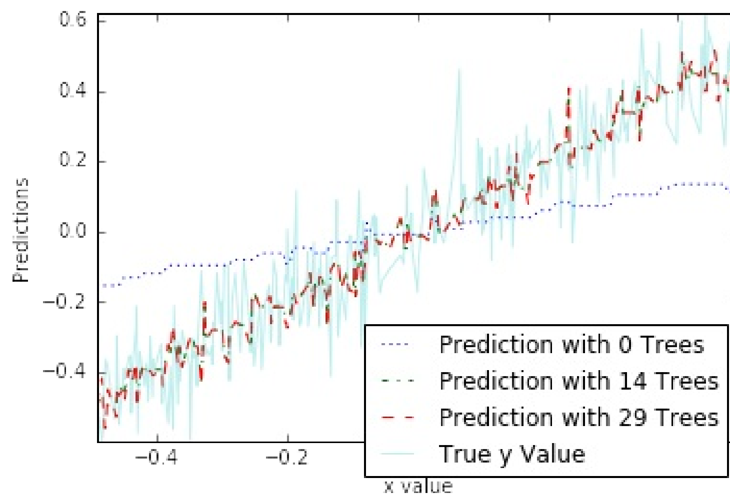
    all_predictions.append(prediction)
    errors = [(y_test[i] - prediction[i]) for i in range(len(y_t
est))]
    mse.append(sum([e * e for e in errors]) / len(y_test))

n_models = [i + 1 for i in range(len(mode_list))]
```

```
plot.plot(n_models, mse)
plot.axis('tight')
plot.xlabel('Number of Models in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()

plot_list = [0, 14, 29]
line_type = [':', '-.', '--']
plot.figure()
for i in range(len(plot_list)):
    i_plot = plot_list[i]
    text_legend = 'Prediction with ' + str(i_plot) + ' Trees'
    plot.plot(x_test, all_predictions[i_plot], label=text_legend
,
                    linestyle=line_type[i])
plot.plot(x_test, y_test, label='True y Value', alpha=0.25)
plot.legend(bbox_to_anchor=(1, 0.3))
plot.axis('tight')
plot.xlabel('x value')
plot.ylabel('Predictions');
```





随机森林算法详解

```
import urllib2
import numpy
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
import random
from math import sqrt
import matplotlib.pyplot as plot

# read data into iterable
target_url = "ftp://ftp.ics.uci.edu/pub/machine-learning-databas
es/wine-quality/winequality-red.csv"
data = urllib2.urlopen(target_url)

x_list = []
labels = []
names = []
first_line = True
for line in data:
    if first_line:
        names = line.strip().split(";")
        first_line = False
    else:
        # split on semi-colon
```

```
        row = line.strip().split(";")
        # put labels in separate array
        labels.append(float(row[-1]))
        # remove label from row
        row.pop()
        # convert row to floats
        float_row = [float(num) for num in row]
        x_list.append(float_row)

nrows = len(x_list)
ncols = len(x_list[0])

# take fixed test set 30% of sample
random.seed(1)  # set seed so results are the same each run
n_sample = int(nrows * 0.30)
idx_test = random.sample(range(nrows), n_sample)
idx_test.sort()
idx_train = [idx for idx in range(nrows) if not (idx in idx_test
)]

# Define test and training attribute and label sets
x_train = [x_list[r] for r in idx_train]
x_test = [x_list[r] for r in idx_test]
y_train = [labels[r] for r in idx_train]
y_test = [labels[r] for r in idx_test]

# train a series of models on random subsets of the training data

# collect the models in a list and check error of composite as l
ist grows

# maximum number of models to generate
num_trees_max = 30

# tree depth - typically at the high end
tree_depth = 12

# pick how many attributes will be used in each model.
# authors recommend 1/3 for regression problem
n_attr = 4
```

```
# initialize a list to hold models
mode_list = []
index_list = []
pred_list = []
n_train_rows = len(y_train)

for i_trees in range(num_trees_max):

    mode_list.append(DecisionTreeRegressor(max_depth=tree_depth)
)

    # take random sample of attributes
    idx_attr = random.sample(range(ncols), n_attr)
    idx_attr.sort()
    index_list.append(idx_attr)

    # take a random sample of training rows
    idx_rows = []
    for i in range(int(0.5 * n_train_rows)):
        idx_rows.append(random.choice(range(len(x_train))))
    idx_rows.sort()

    # build training set
    x_rf_train = []
    y_rf_train = []

    for i in range(len(idx_rows)):
        temp = [x_train[idx_rows[i]][j] for j in idx_attr]
        x_rf_train.append(temp)
        y_rf_train.append(y_train[idx_rows[i]])

    mode_list[-1].fit(x_rf_train, y_rf_train)

    # restrict xTest to attributes selected for training
    x_rf_test = []
    for xx in x_test:
        temp = [xx[i] for i in idx_attr]
        x_rf_test.append(temp)
```

```
    latest_out_sample_prediction = mode_list[-1].predict(x_rf_te
st)
    pred_list.append(list(latest_out_sample_prediction))

# build cumulative prediction from first "n" models
mse = []
all_predictions = []
for i_models in range(len(mode_list)):

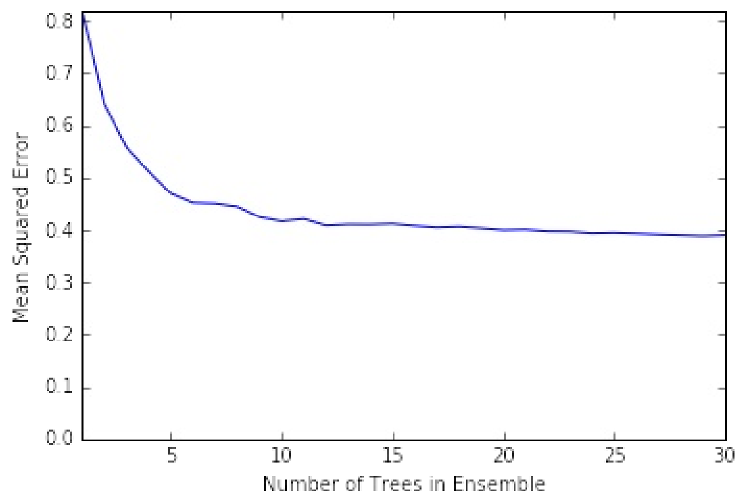
    # add the first "iModels" of the predictions and multiply by
 eps
    prediction = []
    for i_pred in range(len(x_test)):
        prediction.append(
            sum([pred_list[i][i_pred] for i in range(i_models + 1
)]) / (
            i_models + 1))

    all_predictions.append(prediction)
    errors = [(y_test[i] - prediction[i]) for i in range(len(y_t
est))]
    mse.append(sum([e * e for e in errors]) / len(y_test))

n_models = [i + 1 for i in range(len(mode_list))]

plot.plot(n_models, mse)
plot.axis('tight')
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()

print('Minimum MSE')
print(min(mse))
```




```
Minimum MSE
0.389088116065
```

练习题：使用提升树和随机森林两种方法，对信贷数据集进行建模比较

```

```

Sections

What is Feature Engineering?

[back to top]

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.

Sub-Problems of Feature Engineering

- Feature Importance: An estimate of the usefulness of a feature
- Feature Selection: From many features to a few that are useful
- Feature Extraction: The automatic construction of new features from raw data
- Feature Construction: The manual construction of new features from raw data

Iterative Process of Feature Engineering

- Brainstorm features: Really get into the problem, look at a lot of data, study feature engineering on other problems and see what you can steal.
- Devise features: Depends on your problem, but you may use automatic feature extraction, manual feature construction and mixtures of the two.
- Select features: Use different feature importance scorings and feature selection methods to prepare one or more “views” for your models to operate upon.
- Evaluate models: Estimate model accuracy on unseen data using the chosen features.

General Examples of Feature Engineering

- Decompose Categorical Attributes
    - Imagine you have a categorical attribute, like “Item_Color” that can be Red, Blue or Unknown.
- Decompose a Date-Time
    - A date-time contains a lot of information that can be difficult for a model to take advantage of in it’s native form, such as ISO 8601 (i.e. 2014-09-20T20:45:40Z).
- Reframe Numerical Quantities
    - Your data is very likely to contain quantities, which can be reframed to better expose relevant structures. This may be a transform into a new unit or the decomposition of a rate into time and amount components.

Data preprocessing

Dealing with missing data

[back to top]

```
# 构造含缺失值的数据, NaN 表示 Not a Number
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(1, 13).reshape(3, 4),
                  columns=['A', 'B', 'C', 'D'])

df.loc[1, 'C'] = None
df.loc[2, 'D'] = None


df
```

	A	B	C	D
0	1	2	3.0	4.0
1	5	6	NaN	8.0
2	9	10	11.0	NaN

```
# isnull 会返回一个 DataFrame, 里面的 bool 值表示原始数据是否缺失
df.isnull()
```

	A	B	C	D
0	False	False	False	False
1	False	False	True	False
2	False	False	False	True

```
# 结果显示 A 和 B 列没有缺失值, C 和 D 各有一个缺失值
df.isnull().sum()
```

```
A    0
B    0
C    1
D    1
dtype: int64
```

Eliminating samples or features with missing values

处理缺失值最简单的方法就是删掉有缺失的行或者列

[back to top]

```
df.dropna() # 默认删除行 axis = 0
```

	A	B	C	D
0	1	2	3.0	4.0

```
df.dropna(axis=1) # 删除列
```

	A	B
0	1	2
1	5	6
2	9	10

```
# 只删除全是缺失值的行
df.dropna(how='all')
```

	A	B	C	D
0	1	2	3.0	4.0
1	5	6	NaN	8.0
2	9	10	11.0	NaN

```
# 删除非缺失值少于 thresh 的行
df.dropna(thresh=4)
```

	A	B	C	D
0	1	2	3.0	4.0

```
# 删除有缺失值出现在特定列的行
df.dropna(subset=['C'])
```

	A	B	C	D
0	1	2	3.0	4.0
2	9	10	11.0	NaN

看上去删除是很简便的处理方法，但实际上直接删除可能会丢失不少信息，更好的选择是填补缺失值

Imputing missing values

估计缺失值并填充，最普遍的是 mean imputation，也就是用平均值填充

[back to top]

```
from sklearn.preprocessing import Imputer

imr = Imputer(missing_values='NaN', strategy='mean', axis=0)
# If axis=0, then impute along columns.
# If axis=1, then impute along rows.
imr = imr.fit(df.values)
imputed_data = imr.transform(df.values)
imputed_data
```

```
array([[  1.,   2.,   3.,   4.],
       [  5.,   6.,   7.,   8.],
       [  9.,  10.,  11.,   6.]])
```

```
df.values # 并没有改变原先的 df
```

```
array([[  1.,   2.,   3.,   4.],
       [  5.,   6.,  nan,   8.],
       [  9.,  10.,  11.,  nan]])
```

Handling categorical data

对 categorical 需要区分 nominal 和 ordinal 两种类型, nominal 是无序的, 而 ordinal 是有序的

[back to top]

```
import pandas as pd
df = pd.DataFrame([
            ['green', 'M', 10.1, 'class1'],
            ['red', 'L', 13.5, 'class2'],
            ['blue', 'XL', 15.3, 'class1']])

df.columns = ['color', 'size', 'price', 'classlabel']
df
```

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class2
2	blue	XL	15.3	class1

- `color` : nominal feature
- `size` : ordinal feature, XL > L > M
- `price` : numerical feature

Mapping ordinal features

convert the categorical string values into integers

[back to top]

```
# define the mapping manually
size_mapping = {
           'XL': 3,
           'L': 2,
           'M': 1}

df['size'] = df['size'].map(size_mapping)
df
```

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

```
# transform the integer values back to the original string
inv_size_mapping = {v: k for k, v in size_mapping.items()}
df['size'].map(inv_size_mapping)
```

```
0     M
1     L
2    XL
Name: size, dtype: object
```

Encoding class labels

对应 nominal 的 class labels, 也需要将其转换为数值表征，记住此时的数值只代表一个类别，并不表征数值关系

[back to top]

```
import numpy as np

class_mapping = {label:idx for idx,label in
                 enumerate(np.unique(df['classlabel']))}
class_mapping
```

```
{'class1': 0, 'class2': 1}
```

```
# 最终把 classlabel 也转化为 interger
df['classlabel'] = df['classlabel'].map(class_mapping)
df
```

	color	size	price	classlabel
0	green	1	10.1	0
1	red	2	13.5	1
2	blue	3	15.3	0

```
# 转化回来也是 ok 的
inv_class_mapping = {v: k for k, v in class_mapping.items()}
df['classlabel'] = df['classlabel'].map(inv_class_mapping)
df
```

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

```
# sklearn 中也有相应函数
from sklearn.preprocessing import LabelEncoder

class_le = LabelEncoder()
y = class_le.fit_transform(df['classlabel'].values)
y
```

```
array([0, 1, 0])
```

```
# 同样也可以反向转换
class_le.inverse_transform(y)
```

```
array(['class1', 'class2', 'class1'], dtype=object)
```

Performing one-hot encoding on nominal features

[back to top]

```
X = df[['color', 'size', 'price']].values

# color column
color_le = LabelEncoder()
X[:, 0] = color_le.fit_transform(X[:, 0])
X

#blue 0
#green 1
#red 2
```

```
array([[1, 1, 10.1],
       [2, 2, 13.5],
       [0, 3, 15.3]], dtype=object)
```

虽然 color 转化为了 0, 1, 2, 但并不能直接使用来建模, 因为在实际使用中, 会认为 2 大于 1, 也就是 red 大于 green. 实际却不是这样的, 所以需要用到 one-hot encoding, 需要使用 dummy variable, 每一个 label 最后被表示为一个向量. 例如, blue sample can be encoded as blue=1, green=0, red=0.

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(categorical_features=[0], sparse=False)
# 不设定 sparse=False 的话，onehot 会返回一个 sparse matrix， 可以用
toarray() 将之变回 dense

ohe.fit_transform(X)
# 前三列为dummy
```

```
array([[  0. ,   1. ,   0. ,   1. ,  10.1],
       [  0. ,   0. ,   1. ,   2. ,  13.5],
       [  1. ,   0. ,   0. ,   3. ,  15.3]])
```

```
# pandas 中的 get_dummies 函数是生成 dummy variable 更简单的方法
pd.get_dummies(df[['price', 'color', 'size']])
```

	price	size	color_blue	color_green	color_red
0	10.1	1	0.0	1.0	0.0
1	13.5	2	0.0	0.0	1.0
2	15.3	3	1.0	0.0	0.0

Partitioning a dataset in training and test sets

the test set can be understood as the `ultimate test` of our model before we let it loose on the real world

[back to top]

```
# 读取wine数据
df_wine = pd.read_csv('data/wine.data', header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash'
,
'Alcalinity of ash', 'Magnesium', 'Total phenols',
'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proli
ne']

print('Class labels', np.unique(df_wine['Class label']))
df_wine.head()

# 一共有三种 label
```

```
('Class labels', array([1, 2, 3]))
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols
0	1	14.23	1.71	2.43	15.6	127	2.80
1	1	13.20	1.78	2.14	11.2	100	2.65
2	1	13.16	2.36	2.67	18.6	101	2.80
3	1	14.37	1.95	2.50	16.8	113	3.85
4	1	13.24	2.59	2.87	21.0	118	2.80

使用 train_test_split 函数进行训练/测试集切分

```
from sklearn.cross_validation import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=0.3, random_state=0)

# 30%是 test data
```

stratified train test split

stratified 切分，使切分后的数据集更好地保留标签的相对比例

```
# 帮助函数，计算各标签比例
def label_frequency(labels):
    counts = np.unique(labels, return_counts=True)[1]
    n = len(labels)
    return counts / float(n)
```

```
# 原始数据中各标签的比例
label_frequency(y)
```

```
array([ 0.33146067,  0.3988764 ,  0.26966292])
```

```
# train_test_split 后的比例
label_frequency(y_train), label_frequency(y_test)
```

```
(array([ 0.32258065,  0.39516129,  0.28225806]),
 array([ 0.35185185,  0.40740741,  0.24074074]))
```

```
# stratified 之后的标签比例， 更接近原始比例
X_train, X_test, y_train, y_test = \
        train_test_split(X, y, stratify=y, test_size=0.3, random
_state=0)
label_frequency(y_train), label_frequency(y_test)
```

```
(array([ 0.33333333,  0.39837398,  0.26829268]),
 array([ 0.32727273,  0.4       ,  0.27272727]))
```

Bringing features onto the same scale

Feature Scaling 很容易被遗忘, 虽然在 Decision tree和 random forests 时不用担心这个问题. 但在很多算法和模型下都是 scaling 后拟合效果更好.

两类常用方法: normalization 和 standardization.

- normalization: rescaling to [0,1], 如 min-max scaling $x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$
- standardization: more practical, 因为在一些算法中, weights 初始值都设置为 0, 或者接近 0. standardization 之后会更利用更新 weights. 并且 standardize 对 outlier 更不敏感，受影响更小 $x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$

[back to top]

```
# min-max rescaling
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)  # 注意测试集是按照训练集的参数进
行转换
```

```
# standarzation
from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
```

A visual example:

```
ex = pd.DataFrame([0, 1, 2 ,3, 4, 5])

# standardize
ex[1] = (ex[0] - ex[0].mean()) / ex[0].std()
# normalize
ex[2] = (ex[0] - ex[0].min()) / (ex[0].max() - ex[0].min())
ex.columns = ['input', 'standardized', 'normalized']
ex
```

	input	standardized	normalized
0	0	-1.336306	0.0
1	1	-0.801784	0.2
2	2	-0.267261	0.4
3	3	0.267261	0.6
4	4	0.801784	0.8
5	5	1.336306	1.0

Feature selection

Often we collected many features that might be related to a supervised prediction task, but we don't know which of them are actually predictive. To improve interpretability, and sometimes also generalization performance, we can use

feature selection to select a subset of the original features.

[back to top]

根据 John, Kohavi, and Pfleger (1994)，可将特征选择的方法分为两类:

- Wrapper methods evaluate multiple models using procedures that add and/or remove predictors to find the optimal combination that maximizes model performance. In essence, wrapper methods are search algorithms that treat the predictors as the inputs and utilize model performance as the output to be optimized.
- Filter methods evaluate the relevance of the predictors outside of the predictive models and subsequently model only the predictors that pass some criterion. For example, for classification problems, each predictor could be individually evaluated to check if there is a plausible relationship between it and the observed classes. Only predictors with important relationships would then be included in a classification model. Saeys, Inza, and Larranaga (2007) surveys filter methods.

Both approaches have advantages and drawbacks. Filter methods are usually more computationally efficient than wrapper methods, but the selection criterion is not directly related to the effectiveness of the model. Also, most filter methods evaluate each predictor separately and, consequently, redundant (i.e. highly-correlated) predictors may be selected and important interactions between variables will not be able to be quantified. The downside of the wrapper method is that many models are evaluated (which may also require parameter tuning) and thus an increase in computation time. There is also an increased risk of over-fitting with wrappers.

Sklearn 中主要使用 Filter methods. 下面将介绍如何用 sklearn 进行特征选择。

Univariate statistics

[back to top]

The simplest method to select features is using univariate statistics, that is by looking at each feature individually and running a statistical test to see whether it is related to the target.

sklearn 中可以用到的 Univariate statistics 有：

- for regression: f_regression
- for classification: chi2 or f_classif

得到统计量和 p 值之后，sklearn 又配套了不同的选择方法：

- SelectKBest removes all but the k highest scoring features
- SelectPercentile removes all but a user-specified highest scoring percentage of features
- using common univariate statistical tests for each feature: false positive rate SelectFpr, false discovery rate SelectFdr, or family wise error SelectFwe.
- GenericUnivariateSelect allows to perform univariate feature selection with a configurable strategy. This allows to select the best univariate selection strategy with hyper-parameter search estimator.

```
# 以 chi2 和 SelectKbest 为例
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest

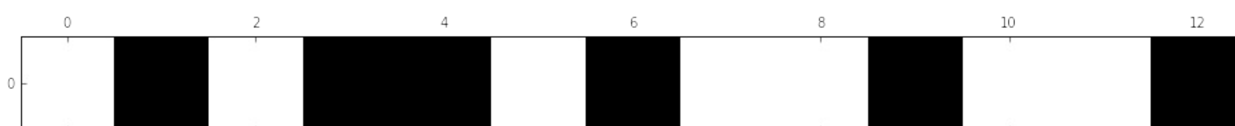
select = SelectKBest(chi2, k=6)
X_uni_selected = select.fit_transform(X_train, y_train)

print(X_train.shape)
print(X_uni_selected.shape)
```

```
(123, 13)
(123, 6)
```

```
import matplotlib.pyplot as plt
%matplotlib inline
# 查看选出了哪几个 feature，黑色是选出来的
mask = select.get_support()
print(mask)
# visualize the mask. black is True, white is False
plt.matshow(mask.reshape(1, -1), cmap='gray_r');
```

```
[False  True False  True  True False  True False False  True Fal
se False
  True]
```



Recursive feature elimination

[back to top]

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and weights are assigned to each one of them. Then, features whose absolute weights are the smallest are pruned from the current set features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

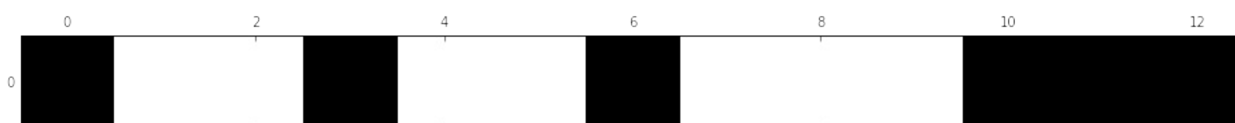
```
from sklearn.feature_selection import RFE
from sklearn.svm import SVC

svc = SVC(kernel="linear", C=1)
rfe = RFE(estimator=svc,
          n_features_to_select=6,  # 要选出几个 feature
          step=1)  # 每次剔除出几个feature
rfe.fit(X_train_std, y_train)

X_rfe_selected = rfe.transform(X_train_std)
```

```
# 查看选出了哪几个 feature
mask = rfe.get_support()
print(mask)
plt.matshow(mask.reshape(1, -1), cmap='gray_r');
```

```
[ True False False  True False False  True False False False  Tr
ue  True
  True]
```



Feature selection using SelectFromModel

[back to top]

SelectFromModel is a meta-transformer that can be used along with any estimator that has a coef *or feature_importances* attribute after fitting. The features are considered unimportant and removed, if the corresponding coef *or feature_importances* values are below the provided threshold parameter. Apart

from specifying the threshold numerically, there are build-in heuristics for finding a threshold using a string argument. Available heuristics are “mean”, “median” and float multiples of these like “0.1*mean”.

一些模型能比较每个 feature 的重要程度，例如 线性模型加上 L1 正则项之后不重要的特征的系数会惩罚为0，随机森林模型能计算每个 feature 的重要程度。
然后 sklearn 有个 SelectFromModel 函数可以配合这些模型进行特征选择

L1-based feature selection

- L2 norm: $||w||_2^2 = \sum_{j=1}^{m} w_j^2$
- L1 norm: $||w||_1 = \sum_{j=1}^{m} |w_j|$
    - 与 L2 正则相比，L1 正则会让更多系数为 0
    - 如果有个高维数据, 有很多特征是无用的, 那么 L1 regularization 就可以被当做一种特征选择的方法.

[back to top]

```
from sklearn.linear_model import LogisticRegression
# sklearn 里想用 L1 正则，把 penalty 参数设为 'l1' 即可
lr = LogisticRegression(penalty='l1', C=0.1)
lr.fit(X_train_std, y_train)
print('Training accuracy:', lr.score(X_train_std, y_train))
print('Test accuracy:', lr.score(X_test_std, y_test))
```

```
('Training accuracy:', 0.98373983739837401)
('Test accuracy:', 0.96363636363636362)
```

加上 L1 正则项后，训练集和测试集上的表现相近，没有过拟合

```
lr.intercept_
```

```
array([-0.26943618, -0.12656436, -0.79402866])
```

```
# 用了 One-vs-Rest (OvR) 方法，所以会出现三行系数
lr.coef_
```

```
array([[ 0.18750685,  0.        ,  0.        ,  0.        ,  0.
       ,
         0.        ,  0.56622652,  0.        ,  0.        ,  0.
       ,
         0.        ,  0.        ,  1.60382013],
       [-0.74867392, -0.04330592, -0.00242426,  0.        ,  0.
       ,
         0.        ,  0.        ,  0.        ,  0.        , -0.8
0946123,
         0.        ,  0.04873335, -0.44621713],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  0.
       ,
         0.        , -0.7299406 ,  0.        ,  0.        ,  0.4
2356047,
        -0.33037171, -0.52828297,  0.        ]])
```

可以看出系数矩阵是稀疏的 (只有少数非零系数)

```
# weights coeff of the different features for different regulari
zation strengths
import matplotlib.pyplot as plt
%matplotlib inline

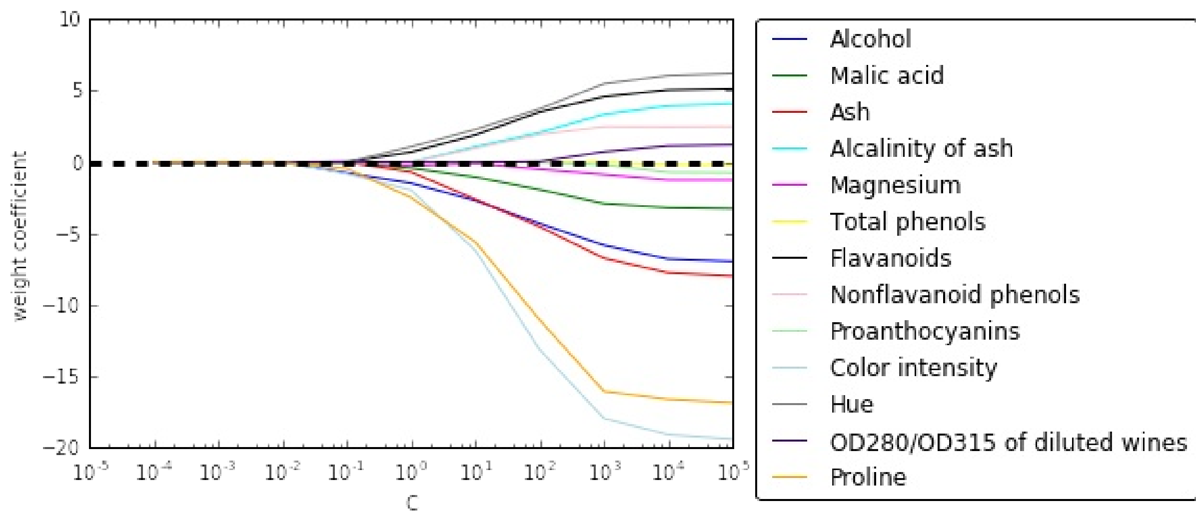
fig = plt.figure()
ax = plt.subplot(111)

colors = ['blue', 'green', 'red', 'cyan',
         'magenta', 'yellow', 'black',
          'pink', 'lightgreen', 'lightblue',
          'gray', 'indigo', 'orange']

weights, params = [], []
for c in np.arange(-4, 6):
    lr = LogisticRegression(penalty='l1', C=10**c, random_state=0
)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)

for column, color in zip(range(weights.shape[1]), colors):
    plt.plot(params, weights[:, column],
             label=df_wine.columns[column+1],
             color=color)
plt.axhline(0, color='black', linestyle='--', linewidth=3)
plt.xlim([10**(-5), 10**5])
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.xscale('log')
plt.legend(loc='upper left')
ax.legend(loc='upper center',
          bbox_to_anchor=(1.38, 1.03),
          ncol=1, fancybox=True);
# plt.savefig('./figures/l1_path.png', dpi=300)
```



随着 L1 正则项增大，无关特征别排除出模型 (系数变为 0)，因此 L1 正则可以作为特征选择的一种方法

结合 sklearn 的 SelectFromModel 进行选择

```
from sklearn.feature_selection import SelectFromModel

model_l1 = SelectFromModel(lr, threshold='median', prefit=True)
X_l1_selected = model_l1.transform(X)
```

```
# 查看选出了哪几个 feature, 黑色是选出来的
mask = model_l1.get_support()
print(mask)
plt.matshow(mask.reshape(1, -1), cmap='gray_r');
```

```
[ True False  True  True False False  True False False  True  Tr
ue False
  True]
```



Tree-based feature selection

随机森林算法可以测量各个特征的重要性，因此可以作为特征选择的一种手段

[back to top]

```
from sklearn.ensemble import RandomForestClassifier

feat_labels = df_wine.columns[1:]

# 使用 decision tree 或 random forests 不需要 standardization或 no
rmalization
forest = RandomForestClassifier(n_estimators=1000,
                                random_state=0,
                                n_jobs=-1)
forest.fit(X_train, y_train)

# random forest 比较特殊, 有 feature_importances 这个 attribute
importances = forest.feature_importances_

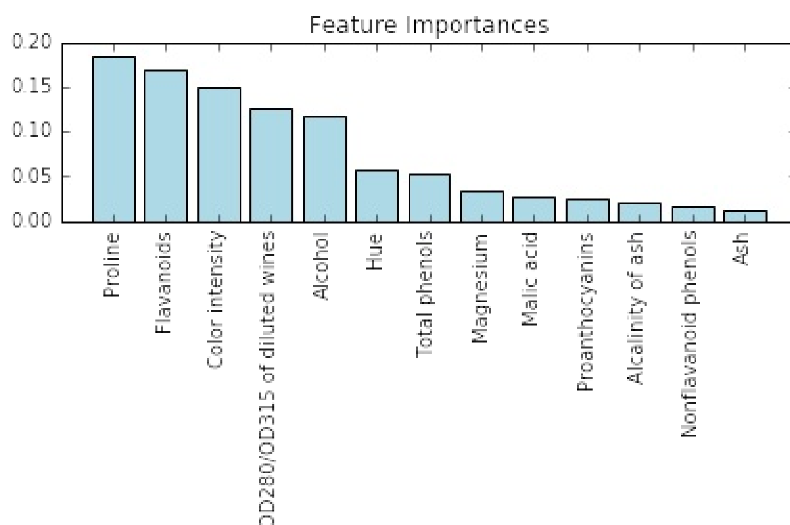
indices = np.argsort(importances)[::-1]
```

```
for i, idx in enumerate(indices):
    print("%2d) %-*s %f" % (i + 1, 30,
                            feat_labels[idx],
                            importances[idx]))
```

```
 1) Proline                        0.185412
 2) Flavanoids                     0.169830
 3) Color intensity                0.149659
 4) OD280/OD315 of diluted wines   0.127238
 5) Alcohol                        0.117432
 6) Hue                            0.057148
 7) Total phenols                  0.053042
 8) Magnesium                      0.034654
 9) Malic acid                     0.027965
10) Proanthocyanins                0.025731
11) Alcalinity of ash              0.021699
12) Nonflavanoid phenols           0.017372
13) Ash                            0.012818
```

```
plt.title('Feature Importances')
plt.bar(range(X_train.shape[1]),
        importances[indices],
        color='lightblue',
        align='center')

plt.xticks(range(X_train.shape[1]),
           feat_labels[indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.tight_layout()
#plt.savefig('./random_forest.png', dpi=300)
```



结合 Sklearn 的 SelectFromModel 进行特征选择

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

select_rf = SelectFromModel(forest, threshold=0.1, prefit=True)

# 或者重新训练一个模型
# select = SelectFromModel(RandomForestClassifier(n_estimators=1
0000, random_state=0, n_jobs=-1), threshold=0.15, prefit=True)
# select.fit(X_train, y_train)

X_train_rf = select_rf.transform(X_train)

print(X_train.shape[1])  # 原始特征维度
print(X_train_rf.shape[1])  # 特征选择后特征维度
```

```
13
5
```

```
# 查看选出的特征
mask = select_rf.get_support()
for f in feat_labels[mask]:
    print(f)
```

```
Alcohol
Flavanoids
Color intensity
OD280/OD315 of diluted wines
Proline
```

```
# 可视化特征选择结果，黑色的是选中的，白色的是滤过的
mask = select_rf.get_support()
print(mask)
plt.matshow(mask.reshape(1, -1), cmap='gray_r');
```

```
[ True False False False False False  True False False  True Fal
se  True
  True]
```



也能将随机森林和 Sequential selection 结合起来

```
from sklearn.feature_selection import RFE
select = RFE(RandomForestClassifier(n_estimators=100, random_sta
te=0),
             n_features_to_select=3)

select.fit(X_train, y_train)
# visualize the selected features:
mask = select.get_support()
plt.matshow(mask.reshape(1, -1), cmap='gray_r');
```



Feature extraction

上一节我们学习了 feature selection, 这一节我们要学降维的另一种方法，feature extraction

[back to top]

Unsupervised dimensionality reduction via principal component analysis

- improve computational efficiency
- help to reduce the **curse of dimensionality**
- unsupervised linear transformation technique
- identify patterns in data based on the correlation between features
- PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions that the original one.

summarize PCA algorithm:

1. Standardize the d-dimensional dataset.
2. Construct the covariance matrix.
3. Decompose the covariance matrix into its eigenvectors and eigenvalues.
4. Select k eigenvectors that correspond to the k largest eigenvalues, where k is the dimensionality of the new feature subspace (k ≤ d).
5. Construct a projection matrix W from the "top" k eigenvectors.
6. Transform the d -dimensional input dataset X using the projection matrix W to obtain the new k -dimensional feature subspace.

简单来说，PCA 是在找寻 variance 最大的方向

[back to top]

仍然使用 *Wine* dataset

```
import pandas as pd

df_wine = pd.read_csv('data/wine.data', header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash'
,
'Alcalinity of ash', 'Magnesium', 'Total phenols',
'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proli
ne']

df_wine.head()
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols
0	1	14.23	1.71	2.43	15.6	127	2.80
1	1	13.20	1.78	2.14	11.2	100	2.65
2	1	13.16	2.36	2.67	18.6	101	2.80
3	1	14.37	1.95	2.50	16.8	113	3.85
4	1	13.24	2.59	2.87	21.0	118	2.80

Splitting the data into 70% training and 30% test subsets.

```
from sklearn.cross_validation import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=0.3, random_state=0)
```

Standardizing the data.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

计算协方差矩阵：$\sigma_{jk} = \frac{1}{n}\sum_{i=1}^{n}(x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$ 通过特征分解得到特征值 λ 和特征向量 v

```
import numpy as np
# compute covariance matrix
cov_mat = np.cov(X_train_std.T)

# get eigenvalues and eigenvectors
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
# eigen_vecs 13*13
print('Eigenvalues \n %s' % eigen_vals)
```

```
Eigenvalues
 [ 4.8923083   2.46635032  1.42809973  1.01233462  0.84906459  0
.60181514
  0.52251546  0.08414846  0.33051429  0.29595018  0.16831254  0.
21432212
  0.2399553 ]
```

Total and explained variance

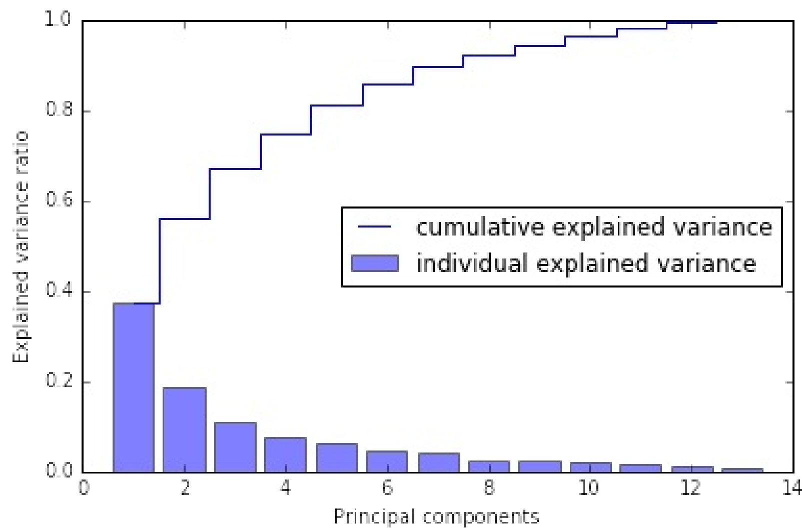
The variance explained ratio of an eigenvalue is simply the fraction of an eigenvalue and the total sum of the eigenvalues: $\frac{\lambda_j}{\sum_{i=1}^{d} \lambda_i}$

[back to top]

```
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)  # cumulative sum of explained
variance
```

```
# plot variance
import matplotlib.pyplot as plt
%matplotlib inline

plt.bar(range(1, 14), var_exp, alpha=0.5, align='center',
        label='individual explained variance')
plt.step(range(1, 14), cum_var_exp, where='mid',
         label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('./figures/pca1.png', dpi=300)
```



第一个 component 能解释将近 40% 的 variance, 前两个 components 能解释近 60%

Feature transformation

[back to top]

```
# Make a list of (eigenvalue, eigenvector) tuples
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in
 range(len(eigen_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eigen_pairs.sort(reverse=True)
```

we only chose two eigenvectors for the purpose of illustration, since we are going to plot the data via a two-dimensional scatter plot later in this subsection.

In practice, the number of principal components has to be determined from a trade-off between computational efficiency and the performance of the classifier.

```
w = np.column_stack([eigen_pairs[0][1], eigen_pairs[1][1]])
print(w)  # 13*2 projection matrix from the top two eigenvectors
```

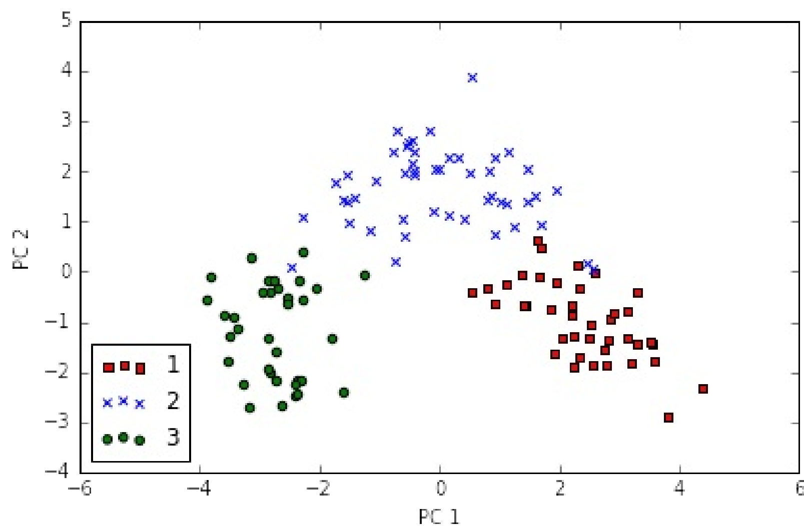
```
[[ 0.14669811 -0.50417079]
 [-0.24224554 -0.24216889]
 [-0.02993442 -0.28698484]
 [-0.25519002  0.06468718]
 [ 0.12079772 -0.22995385]
 [ 0.38934455 -0.09363991]
 [ 0.42326486 -0.01088622]
 [-0.30634956 -0.01870216]
 [ 0.30572219 -0.03040352]
 [-0.09869191 -0.54527081]
 [ 0.30032535  0.27924322]
 [ 0.36821154  0.174365  ]
 [ 0.29259713 -0.36315461]]
```

利用 projection matrix W, 我们可以得到转换后的数据 x' $x' = xW$

```
# transform the entire 124×13-dimensional training dataset onto
the two principal components
X_train_pca = X_train_std.dot(w)
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']

for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train==l, 0],
                X_train_pca[y_train==l, 1],
                c=c, label=l, marker=m)

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('./figures/pca2.png', dpi=300)
```



data is more spread along the x-axis, a linear classier will likely be able to separate the classes well

Principal component analysis in scikit-learn

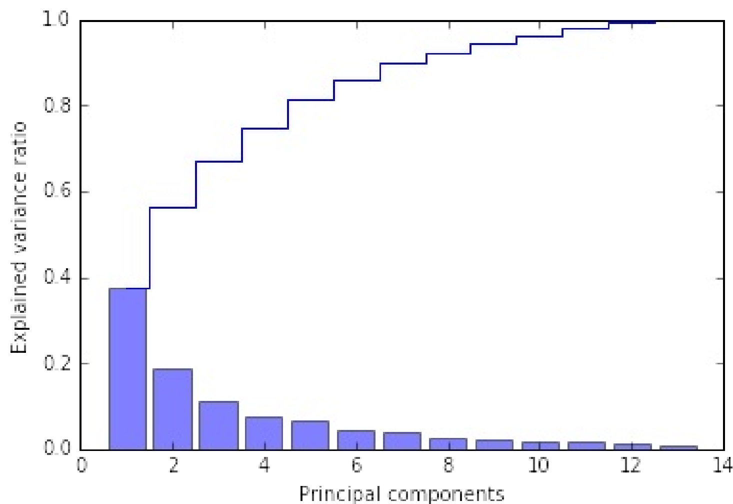
[back to top]

```
from sklearn.decomposition import PCA

pca = PCA()
X_train_pca = pca.fit_transform(X_train_std)
pca.explained_variance_ratio_
```

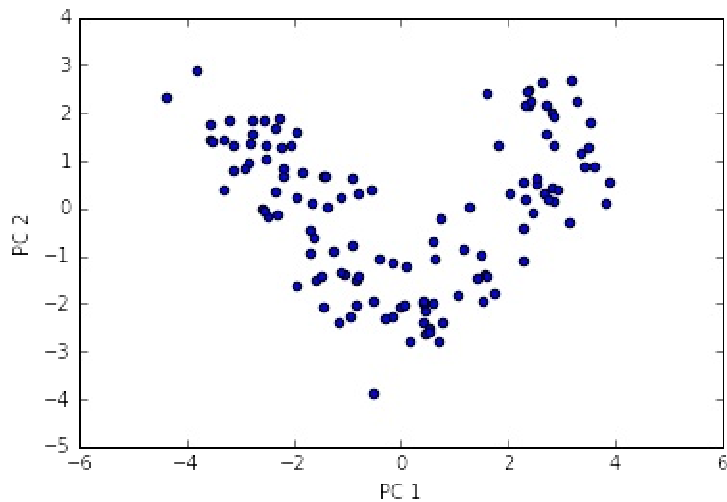
```
array([ 0.37329648,  0.18818926,  0.10896791,  0.07724389,  0.06
478595,
        0.04592014,  0.03986936,  0.02521914,  0.02258181,  0.01
830924,
        0.01635336,  0.01284271,  0.00642076])
```

```
plt.bar(range(1, 14), pca.explained_variance_ratio_, alpha=0.5,
align='center')
plt.step(range(1, 14), np.cumsum(pca.explained_variance_ratio_),
 where='mid')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components');
```




```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
```

```
plt.scatter(X_train_pca[:,0], X_train_pca[:,1])
plt.xlabel('PC 1')
plt.ylabel('PC 2');
```



If we compare the PCA projection via scikit-learn with our own PCA implementation, we notice that the plot above is a mirror image of the previous PCA via our step-by-step approach.
Note that this is not due to an error in any of those two implementations, but the reason for this difference is that, depending on the eigensolver, eigenvectors can have either negative or positive signs.

```
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution)
,
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).
T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

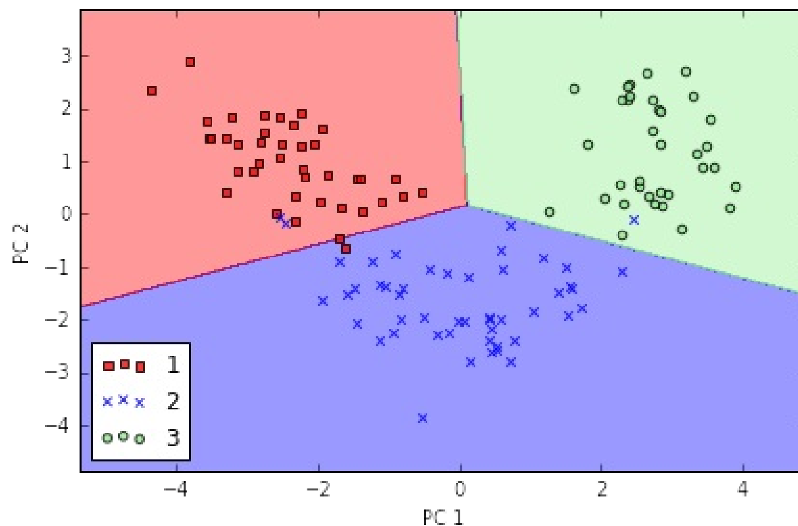
    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)
```

Training logistic regression classifier using the first 2 principal components.

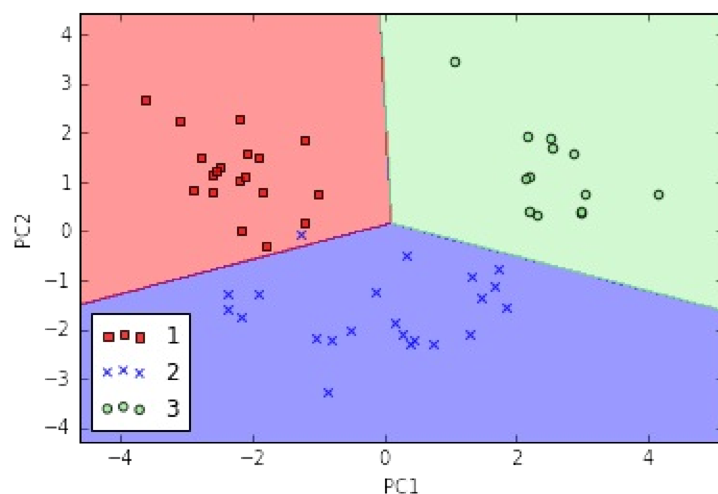
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr = lr.fit(X_train_pca, y_train)
```

```
plot_decision_regions(X_train_pca, y_train, classifier=lr)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout();
# plt.savefig('./figures/pca3.png', dpi=300)
```




```
# 在测试集上测试
plot_decision_regions(X_test_pca, y_test, classifier=lr)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend(loc='lower left');
# 分类效果也很不错
```



Using kernel principal component analysis for nonlinear mappings

via kernel PCA, we perform a nonlinear mapping that transforms the data onto a higher-dimensional space and use standard PCA in this higher-dimensional space to project the data back onto a lower-dimensional space where the samples can be separated by a linear classifier

[back to top]

most commonly used kernel:

- polynomial kernel
- hyperbolic tangent (sigmoid) kernel
- Radial Basis Function (RBF)

to implement RBF kernel PCA:

1. compute the kernel (similarity) matrix k
2. center the kernel matrix k
3. collect the top k eigenvectors of the centered kernel matrix based on their corresponding eigenvalues, ranked by decreasing magnitude.

Implementing a kernel principal component analysis in Python

[back to top]

Radial Basis Function (RBF) or Gaussian kernel: \begin{align} k(x^{(i)}, x^{(j)}) =& exp(-\frac{||x^{(i)} - x^{(j)}||^2}{2\sigma^2}) \ =& exp(-\gamma ||x^{(i)} - x^{(j)}||^2) \end{align}

```
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
import numpy as np

def rbf_kernel_pca(X, gamma, n_components):
    """
    RBF kernel PCA implementation.

    Parameters
    ------------
    X: {NumPy ndarray}, shape = [n_samples, n_features]

    gamma: float
      Tuning parameter of the RBF kernel

    n_components: int
      Number of principal components to return

    Returns
    ------------
     X_pc: {NumPy ndarray}, shape = [n_samples, k_features]
       Projected dataset

    """
    # Calculate pairwise squared Euclidean distances
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')

    # Convert pairwise distances into a square matrix.
    mat_sq_dists = squareform(sq_dists)

    # Compute the symmetric kernel matrix.
    K = exp(-gamma * mat_sq_dists)

    # Center the kernel matrix.
```

```
    N = K.shape[0]
    one_n = np.ones((N,N)) / N
    K = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n
)

    # Obtaining eigenpairs from the centered kernel matrix
    # numpy.eigh returns them in sorted order
    eigvals, eigvecs = eigh(K)

    # Collect the top k eigenvectors (projected samples)
    X_pc = np.column_stack((eigvecs[:, -i]
                            for i in range(1, n_components + 1))
)

    return X_pc
```

Example 1: Separating half-moon shapes

[back to top]

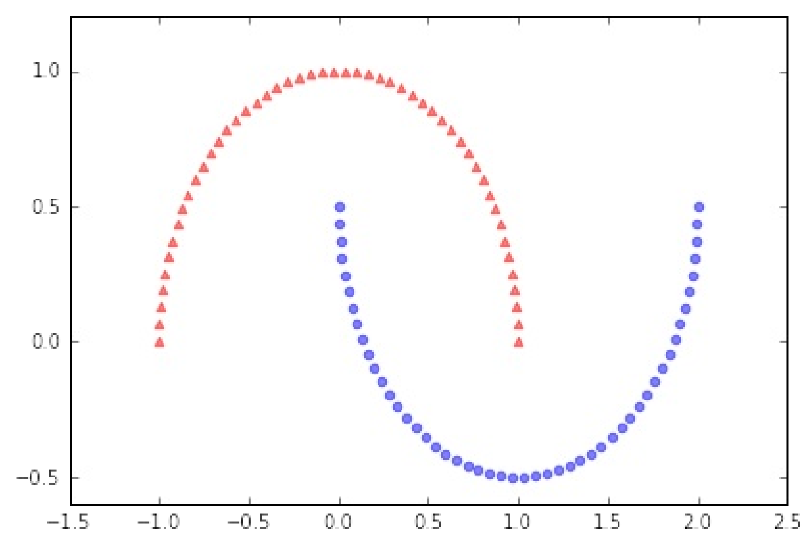
建造月形数据，用以演示

```
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, random_state=123)

plt.scatter(X[y==0, 0], X[y==0, 1], color='red', marker='^', alp
ha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', marker='o', al
pha=0.5)

plt.tight_layout()
# plt.savefig('./figures/half_moon_1.png', dpi=300)
```


1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

```
# standardize PCA
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_std = scaler.fit_transform(X)

scikit_pca = PCA(n_components=2)
X_spca = scikit_pca.fit_transform(X_std)

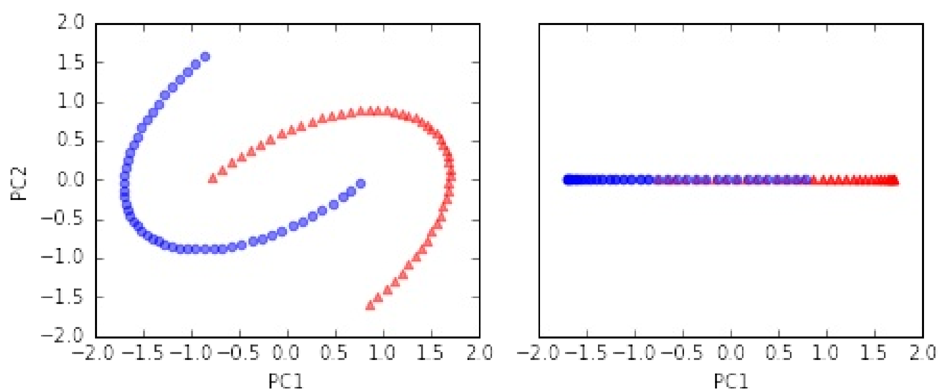
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))

ax[0].scatter(X_spca[y==0, 0], X_spca[y==0, 1],
            color='red', marker='^', alpha=0.5)
ax[0].scatter(X_spca[y==1, 0], X_spca[y==1, 1],
            color='blue', marker='o', alpha=0.5)

ax[1].scatter(X_spca[y==0, 0], np.zeros((50,1)),
            color='red', marker='^', alpha=0.5)
ax[1].scatter(X_spca[y==1, 0], np.zeros((50,1)),
            color='blue', marker='o', alpha=0.5)

ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')

plt.tight_layout()
# plt.savefig('./figures/half_moon_2.png', dpi=300)
```



a linear classier would not be able to perform well

```
# kernel PCA function rbf_kernel_pca
from matplotlib.ticker import FormatStrFormatter

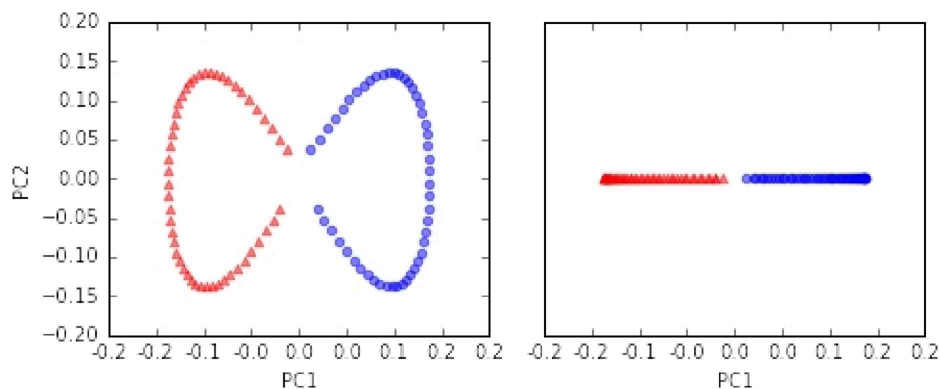
X_kpca = rbf_kernel_pca(X, gamma=15, n_components=2)

fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))
ax[0].scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='^', alpha=0.5)
ax[0].scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='o', alpha=0.5)

ax[1].scatter(X_kpca[y==0, 0], np.zeros((50,1)),
            color='red', marker='^', alpha=0.5)
ax[1].scatter(X_kpca[y==1, 0], np.zeros((50,1)),
            color='blue', marker='o', alpha=0.5)

ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')
ax[0].xaxis.set_major_formatter(FormatStrFormatter('%0.1f'))
ax[1].xaxis.set_major_formatter(FormatStrFormatter('%0.1f'))

plt.tight_layout()
# plt.savefig('./figures/half_moon_3.png', dpi=300)
```



two classes (circles and triangles) are linearly well separated

Example 2: Separating concentric circles

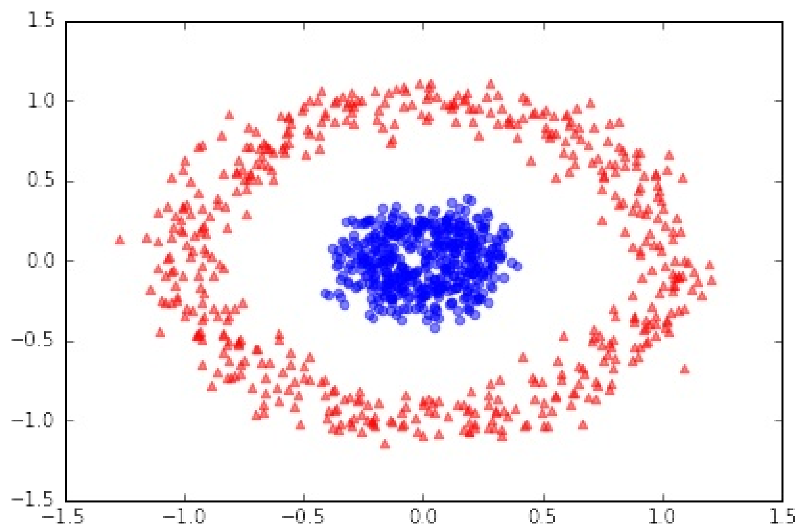
[back to top]

```
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=1000, random_state=123, noise=0.1,
 factor=0.2)

plt.scatter(X[y==0, 0], X[y==0, 1], color='red', marker='^', alp
ha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', marker='o', al
pha=0.5)

plt.tight_layout()
# plt.savefig('./figures/circles_1.png', dpi=300)
```



```
# standard PCA
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

scikit_pca = PCA(n_components=2)
X_spca = scikit_pca.fit_transform(X_std)

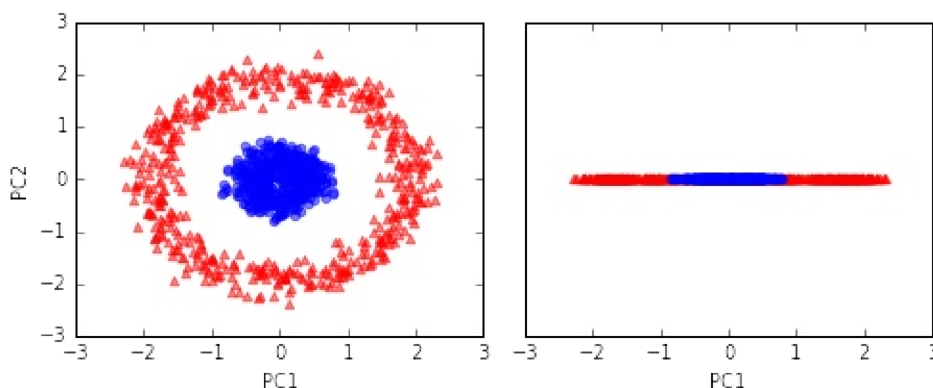
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))

ax[0].scatter(X_spca[y==0, 0], X_spca[y==0, 1],
            color='red', marker='^', alpha=0.5)
ax[0].scatter(X_spca[y==1, 0], X_spca[y==1, 1],
            color='blue', marker='o', alpha=0.5)

ax[1].scatter(X_spca[y==0, 0], np.zeros((500,1)),
            color='red', marker='^', alpha=0.5)
ax[1].scatter(X_spca[y==1, 0], np.zeros((500,1)),
            color='blue', marker='o', alpha=0.5)

ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')

plt.tight_layout()
# plt.savefig('./figures/circles_2.png', dpi=300)
```



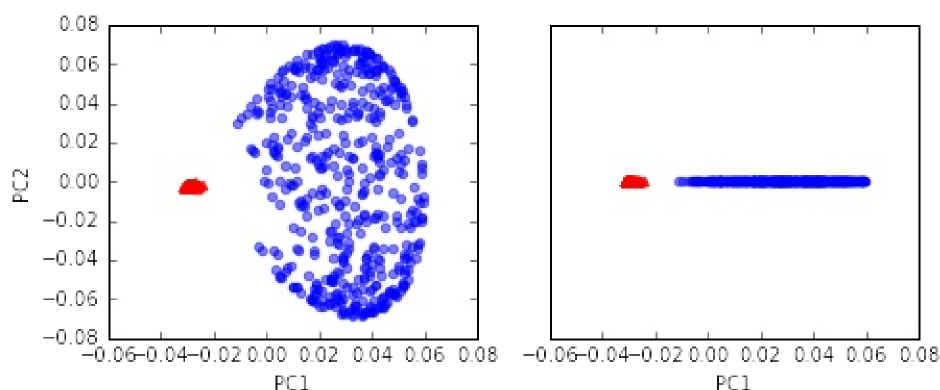
```
# kernel RBF
X_kpca = rbf_kernel_pca(X, gamma=15, n_components=2)

fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))
ax[0].scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='^', alpha=0.5)
ax[0].scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='o', alpha=0.5)

ax[1].scatter(X_kpca[y==0, 0], np.zeros((500,1)),
            color='red', marker='^', alpha=0.5)
ax[1].scatter(X_kpca[y==1, 0], np.zeros((500,1)),
            color='blue', marker='o', alpha=0.5)

ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')

plt.tight_layout()
# plt.savefig('./figures/circles_3.png', dpi=300)
```



Projecting new data points

learn how to project data points that were not part of the training dataset

[back to top]

```
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
import numpy as np

def rbf_kernel_pca(X, gamma, n_components):
    """
    RBF kernel PCA implementation.

    Parameters
    ------------
    X: {NumPy ndarray}, shape = [n_samples, n_features]

    gamma: float
      Tuning parameter of the RBF kernel

    n_components: int
      Number of principal components to return

    Returns
    ------------
     X_pc: {NumPy ndarray}, shape = [n_samples, k_features]
       Projected dataset

     lambdas: list
       Eigenvalues

    """
    # Calculate pairwise squared Euclidean distances
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')

    # Convert pairwise distances into a square matrix.
    mat_sq_dists = squareform(sq_dists)

    # Compute the symmetric kernel matrix.
    K = exp(-gamma * mat_sq_dists)
```

```
    # Center the kernel matrix.
    N = K.shape[0]
    one_n = np.ones((N,N)) / N
    K = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n
)

    # Obtaining eigenpairs from the centered kernel matrix
    # numpy.eigh returns them in sorted order
    eigvals, eigvecs = eigh(K)

    # Collect the top k eigenvectors (projected samples)
    alphas = np.column_stack((eigvecs[:,-i] for i in range(1,n_c
omponents+1)))

    # Collect the corresponding eigenvalues
    lambdas = [eigvals[-i] for i in range(1,n_components+1)]

    return alphas, lambdas
```

```
X, y = make_moons(n_samples=100, random_state=123)
alphas, lambdas = rbf_kernel_pca(X, gamma=15, n_components=1)
```

```
x_new = X[25]
x_new
```

```
array([ 1.8713,  0.0093])
```

```
x_proj = alphas[25] # original projection
x_proj
```

```
array([ 0.0788])
```

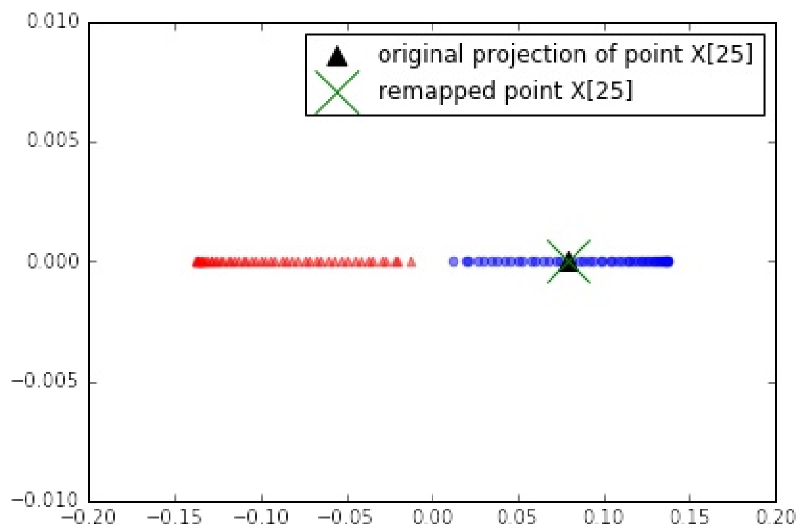
```
def project_x(x_new, X, gamma, alphas, lambdas):
    pair_dist = np.array([np.sum((x_new-row)**2) for row in X])
    k = np.exp(-gamma * pair_dist)
    return k.dot(alphas / lambdas)

# projection of the "new" datapoint
x_reproj = project_x(x_new, X, gamma=15, alphas=alphas, lambdas=
lambdas)
x_reproj
```

```
array([ 0.0788])
```

```
plt.scatter(alphas[y==0, 0], np.zeros((50)),
            color='red', marker='^',alpha=0.5)
plt.scatter(alphas[y==1, 0], np.zeros((50)),
            color='blue', marker='o', alpha=0.5)
plt.scatter(x_proj, 0, color='black', label='original projection
 of point X[25]', marker='^', s=100)
plt.scatter(x_reproj, 0, color='green', label='remapped point X[
25]', marker='x', s=500)
plt.legend(scatterpoints=1)

plt.tight_layout()
# plt.savefig('./figures/reproject.png', dpi=300)
```



project correctly

Kernel principal component analysis in scikit-learn

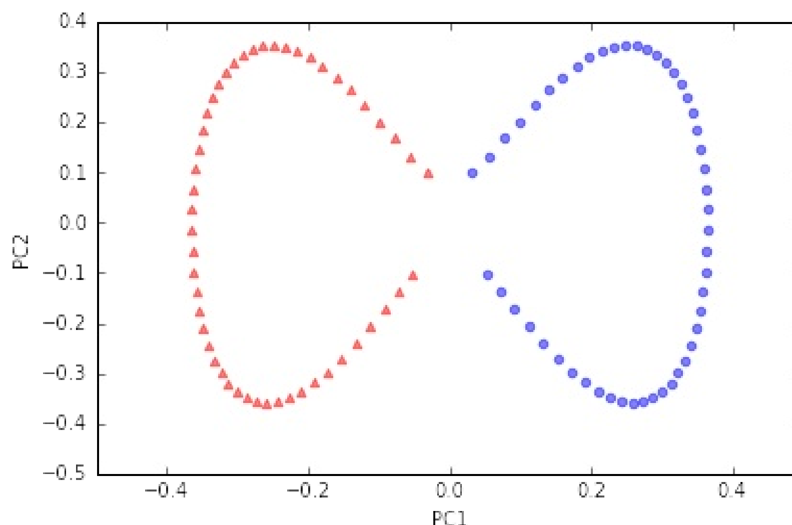
[back to top]

```
from sklearn.decomposition import KernelPCA

X, y = make_moons(n_samples=100, random_state=123)
scikit_kpca = KernelPCA(n_components=2, kernel='rbf', gamma=15)
X_skernpca = scikit_kpca.fit_transform(X)

plt.scatter(X_skernpca[y==0, 0], X_skernpca[y==0, 1],
            color='red', marker='^', alpha=0.5)
plt.scatter(X_skernpca[y==1, 0], X_skernpca[y==1, 1],
            color='blue', marker='o', alpha=0.5)

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.tight_layout()
# plt.savefig('./figures/scikit_kpca.png', dpi=300)
```

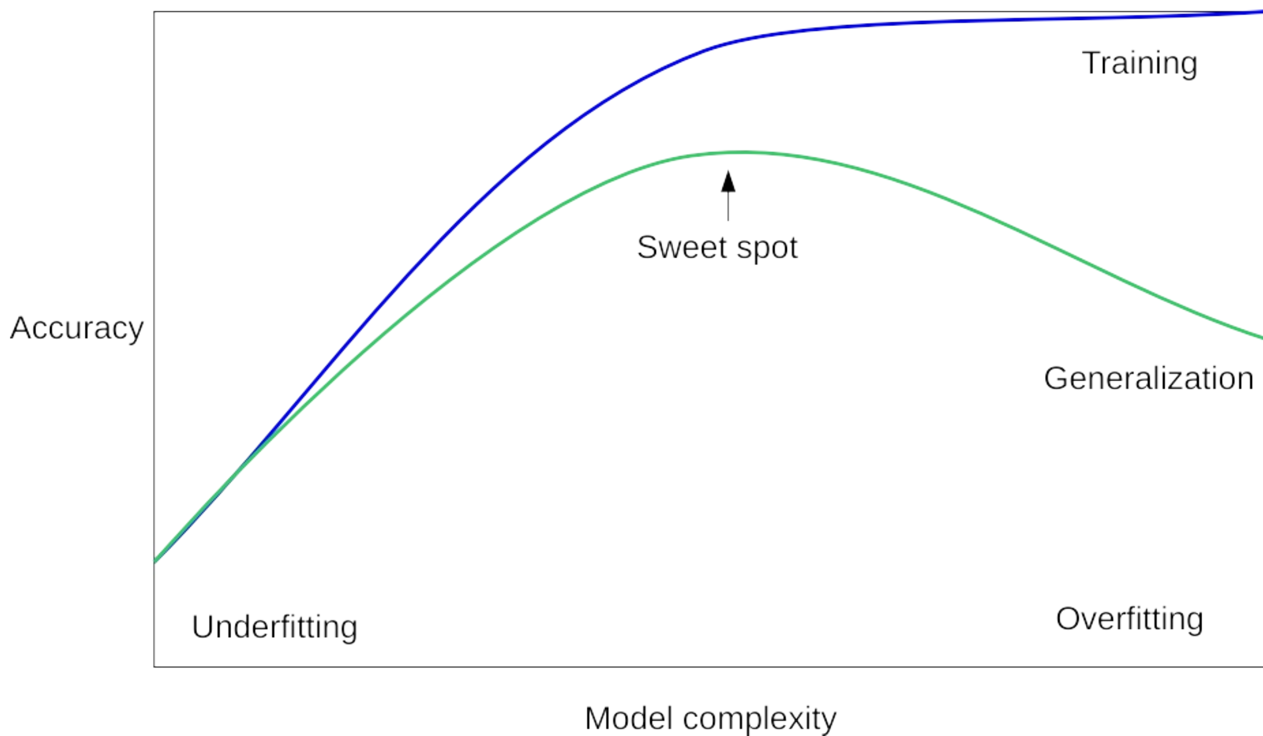


特征工程checklist

- Do you have domain knowledge? If yes, construct a better set of ad hoc”” features
- Are your features commensurate? If no, consider normalizing them.
- Do you suspect interdependence of features? If yes, expand your feature set by constructing conjunctive features or products of features, as much as your computer resources allow you.
- Do you need to prune the input variables (e.g. for cost, speed or data understanding reasons)? If no, construct disjunctive features or weighted sums of feature
- Do you need to assess features individually (e.g. to understand their influence on the system or because their number is so large that you need to do a first filtering)? If yes, use a variable ranking method; else, do it anyway to get baseline results.
- Do you need a predictor? If no, stop
- Do you suspect your data is “dirty” (has a few meaningless input patterns and/or noisy outputs or wrong class labels)? If yes, detect the outlier examples using the top ranking variables obtained in step 5 as representation; check and/or discard them.
- Do you know what to try first? If no, use a linear predictor. Use a forward selection method with the “probe” method as a stopping criterion or use the 0-norm embedded method for comparison, following the ranking of step 5, construct a sequence of predictors of same nature using increasing subsets of features. Can you match or improve performance with a smaller subset? If yes, try a non-linear predictor with that subset.
- Do you have new ideas, time, computational resources, and enough examples? If yes, compare several feature selection methods, including your new idea, correlation coefficients, backward selection and embedded methods. Use linear and non-linear predictors. Select the best approach with model selection
- Do you want a stable solution (to improve performance and/or understanding)? If yes, subsample your data and redo your analysis for several “bootstrap”.

一个使用正则化方法进行变量选择的例子

[back to top]




```
from sklearn import datasets
from sklearn import cross_validation
from sklearn import linear_model
from sklearn import metrics
from sklearn import tree
from sklearn import neighbors
from sklearn import svm
from sklearn import ensemble
from sklearn import cluster

import matplotlib.pyplot as plt
%matplotlib inline

import numpy as np
import seaborn as sns
```

```
np.random.seed(123)
```

```
# 构建 dataset, 50个 sample, 50个 feature
X_all, y_all = datasets.make_regression(n_samples=50, n_features=
50, n_informative=10)
```

```
# 50% train, 50% test
X_train, X_test, y_train, y_test = cross_validation.train_test_s
plit(X_all, y_all, train_size=0.5)
```

```
X_train.shape, y_train.shape
```

```
((25, 50), (25,))
```

```
X_test.shape, y_test.shape
```

```
((25, 50), (25,))
```

Linear Regression

$\min_{w,b} \sum_i ||w^\mathsf{T} x_i + b - y_i||^2$

[back to top]

```
# linear reg
model = linear_model.LinearRegression()
```

```
model.fit(X_train, y_train)
```

```
/Users/alan/anaconda/lib/python2.7/site-packages/scipy/linalg/ba
sic.py:884: RuntimeWarning: internal gelsd driver lwork query er
ror, required iwork dimension not returned. This is likely the r
esult of LAPACK bug 0038, fixed in LAPACK 3.2.2 (released July 2
1, 2010). Falling back to 'gelss' driver.
  warnings.warn(mesg, RuntimeWarning)





LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, norm
alize=False)
```

```
def sse(resid):
    return sum(resid**2)
```

```
# 计算 train data 的 SSE
resid_train = y_train - model.predict(X_train)
sse_train = sse(resid_train)
sse_train
```

```
7.9634561748974877e-25
```

```
# 预测 test 再计算 test data 的SSE
resid_test = y_test - model.predict(X_test)
sse_test = sse(resid_test)
sse_test
```

```
213555.61203039085
```

结果 test data 显示预测效果很差，可能 overfitting

```
model.score(X_train, y_train)
```

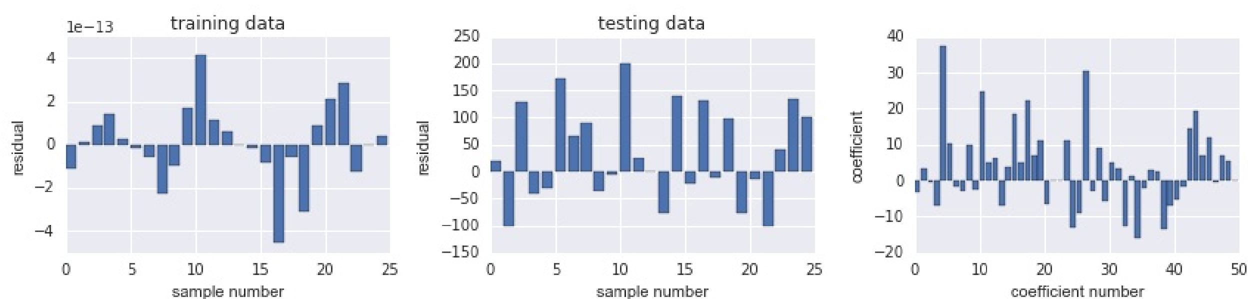
```
1.0
```

```
model.score(X_test, y_test)
```

```
0.31407400675201724
```

```
def plot_residuals_and_coeff(resid_train, resid_test, coeff):
    fig, axes = plt.subplots(1, 3, figsize=(12, 3))
    axes[0].bar(np.arange(len(resid_train)), resid_train)
    axes[0].set_xlabel("sample number")
    axes[0].set_ylabel("residual")
    axes[0].set_title("training data")
    axes[1].bar(np.arange(len(resid_test)), resid_test)
    axes[1].set_xlabel("sample number")
    axes[1].set_ylabel("residual")
    axes[1].set_title("testing data")
    axes[2].bar(np.arange(len(coeff)), coeff)
    axes[2].set_xlabel("coefficient number")
    axes[2].set_ylabel("coefficient")
    fig.tight_layout()
    return fig, axes
```

```
# 画出 residual
fig, ax = plot_residuals_and_coeff(resid_train, resid_test, mode
l.coef_);
```



Ridge Regression

L2 penalized, add squared sum of the weights to least-squares cost function

$$\min_{w,b} \sum_i ||w^\top x_i + b - y_i||^2 + \alpha ||w||_2^2$$

[back to top]

```
# 使用 Ridge 正则化
model = linear_model.Ridge(alpha=5)
```

```
model.fit(X_train, y_train)
```

```
Ridge(alpha=5, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

```
resid_train = y_train - model.predict(X_train)
sse_train = sum(resid_train**2)
sse_train
```

```
3292.9620358692705
```

```
resid_test = y_test - model.predict(X_test)
sse_test = sum(resid_test**2)
sse_test
```

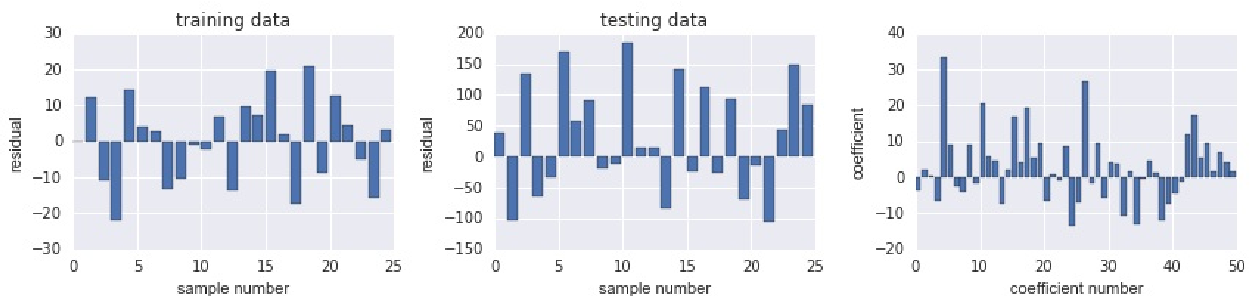
```
209557.58585055024
```

train data的 SSE 提升很多

```
# test model score 仍然不高
model.score(X_train, y_train), model.score(X_test, y_test)
```

```
(0.9900302124332 4718, 0.32691539290134652)
```

```
fig, ax = plot_residuals_and_coeff(resid_train, resid_test, mode
l.coef_)
```



LASSO Regression

L1-norm certain weights can become zero, useful as a supervised feature selection technique.

$\min_{w,b} \sum_i ||w^\mathsf{T} x_i + b - y_i||^2 + \alpha ||w||_1$

[back to top]

```
model = linear_model.Lasso(alpha=1.0)
```

```
model.fit(X_train, y_train)
```

```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_sta
te=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

```
resid_train = y_train - model.predict(X_train)
sse_train = sse(resid_train)
sse_train
```

```
309.74971389532328
```

```
resid_test = y_test - model.predict(X_test)
sse_test = sse(resid_test)
sse_test
```

```
1489.117606500263
```

相较 Ridge, SSE 都减少很多

```
fig, ax = plot_residuals_and_coeff(resid_train, resid_test, mode
l.coef_)
```



上图发现, coeff 有很多都是0

```
alphas = np.logspace(-4, 2, 100)
```

```
# 寻找 LASSO 的最优参数 alpha
coeffs = np.zeros((len(alphas), X_train.shape[1]))
sse_train = np.zeros_like(alphas)
sse_test = np.zeros_like(alphas)

for n, alpha in enumerate(alphas):
    model = linear_model.Lasso(alpha=alpha)
    model.fit(X_train, y_train)
    coeffs[n, :] = model.coef_
    resid = y_train - model.predict(X_train)
    sse_train[n] = sum(resid**2)
    resid = y_test - model.predict(X_test)
    sse_test[n] = sum(resid**2)
```

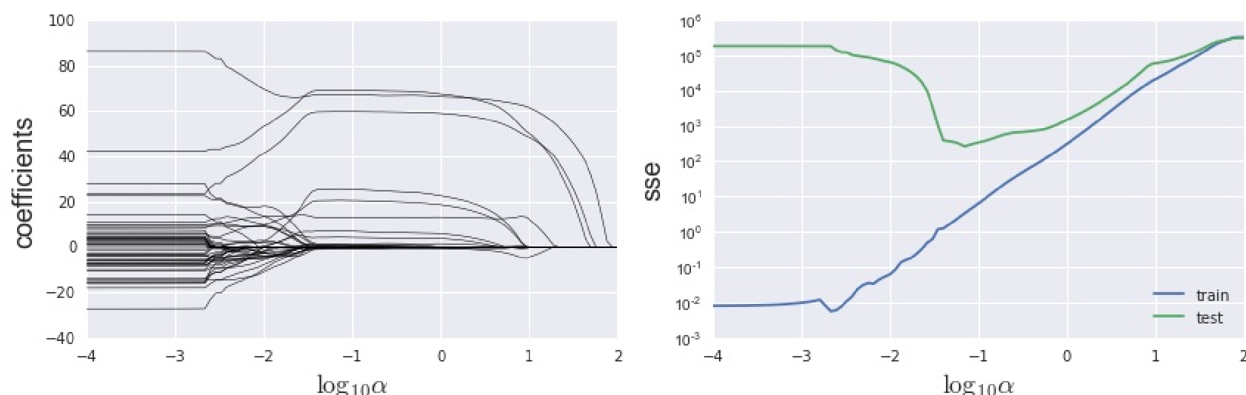
```
/Users/alan/anaconda/lib/python2.7/site-packages/sklearn/linear_
model/coordinate_descent.py:466: ConvergenceWarning: Objective d
id not converge. You might want to increase the number of iterat
ions
  ConvergenceWarning)
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4), sharex=True)

for n in range(coeffs.shape[1]):
    axes[0].plot(np.log10(alphas), coeffs[:, n], color='k', lw=0
.5)

axes[1].semilogy(np.log10(alphas), sse_train, label="train")
axes[1].semilogy(np.log10(alphas), sse_test, label="test")
axes[1].legend(loc=0)

axes[0].set_xlabel(r"${\log_{10}}\alpha$", fontsize=18)
axes[0].set_ylabel(r"coefficients", fontsize=18)
axes[1].set_xlabel(r"${\log_{10}}\alpha$", fontsize=18)
axes[1].set_ylabel(r"sse", fontsize=18)
fig.tight_layout()
```



alpha 越大, coeff 最终都会变成0, 而 train SSE 会先减小再增加, 而 test 是一直在增加.

在-1附近, train SSE 最小, 而 coeff 大概有8个不是0.

```
# 使用LassoCV: Lasso linear model with iterative fitting along a
regularization path
model = linear_model.LassoCV()
```

```
model.fit(X_all, y_all)
```

```
LassoCV(alphas=None, copy_X=True, cv=None, eps=0.001, fit_interc
ept=True,
    max_iter=1000, n_alphas=100, n_jobs=1, normalize=False, posi
tive=False,
    precompute='auto', random_state=None, selection='cyclic', to
l=0.0001,
    verbose=False)
```

```
# 计算出的最佳 alphs
model.alpha_
```

```
0.06559238747534718
```

```
resid_train = y_train - model.predict(X_train)
sse_train = sse(resid_train)
sse_train
```

```
1.5450589323148352
```

```
resid_test = y_test - model.predict(X_test)
sse_test = sse(resid_test)
sse_test
```

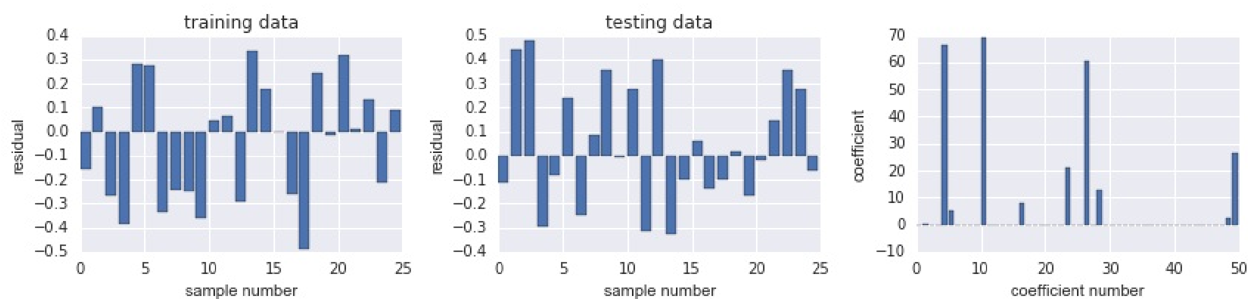
```
1.5321417406216176
```

发现 SSE 都已经比较接近0了

```
model.score(X_train, y_train), model.score(X_test, y_test)
# score 都很高
```

```
(0.99999532217220677, 0.999995078865709 82)
```

```
fig, ax = plot_residuals_and_coeff(resid_train, resid_test, model.coef_)
# 9个 non-zero coeff
```



练习：利用本章学到的方法对信贷数据进行特征工程

(#sections)

Sections

Loading the Breast Cancer Wisconsin dataset

- Breast Cancer Wisconsin 数据包括 569 例良性或恶性癌细胞样本
- 数据前两列是样本 ID 及诊断 (M for malignant 恶性, B for benigh 良性)
- 后面 30 列是细胞核的图片的数据

[back to top]

```
import pandas as pd

df = pd.read_csv('data/wdbc.data', header=None)
df.head()
```

	0	1	2	3	4	5	6	
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.2776
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.0786
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.1599
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.2839
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.1328

5 rows × 32 columns

```
df.shape
```

```
(569, 32)
```

```
# 数据预处理
from sklearn.preprocessing import LabelEncoder
X = df.loc[:, 2:].values
y = df.loc[:, 1].values
le = LabelEncoder()
y = le.fit_transform(y)
le.transform(['M', 'B']) # M->1 B->0
```

```
array([1, 0])
```

```
# 80% train, 20% test
from sklearn.cross_validation import train_test_split

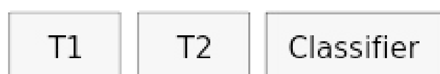
X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=0.20, random_state=1)
```

Streamlining workflows with pipelines

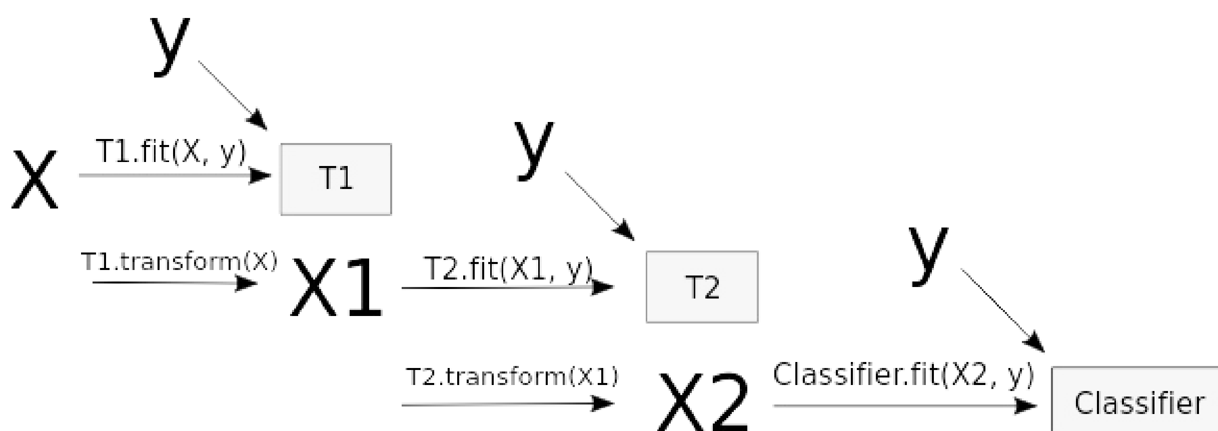
fit a model including an arbitrary number of transformation steps and apply it to make predictions about new data.

[back to top]

pipe = make_pipeline(T1(), T2(), Classifier())



pipe.fit(X, y)



pipe.predict(X)



```
# chain the StandardScaler, PCA, and LogisticRegression objects
in a pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

pipe_lr = Pipeline([('scl', StandardScaler()),  # 标准化原始数据
                    ('pca', PCA(n_components=2)), # PCA 降维
                    ('clf', LogisticRegression(random_state=1))]
)

pipe_lr.fit(X_train, y_train)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
y_pred = pipe_lr.predict(X_test)
```

```
Test Accuracy: 0.947
```

Model evaluation

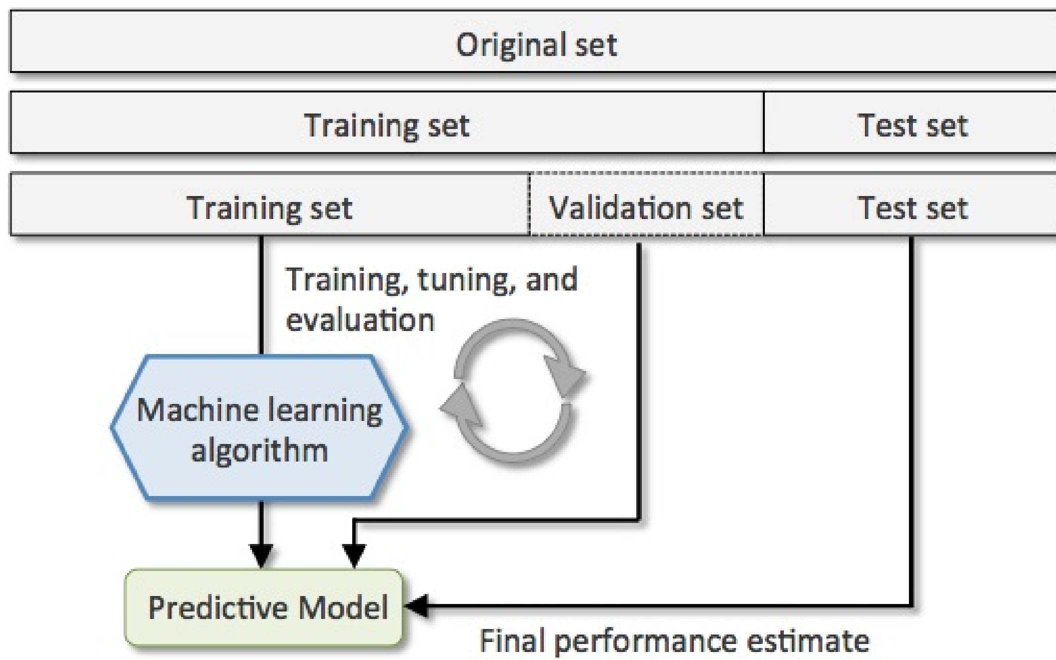
[back to top]

The holdout method

将数据分为三个部分

- training set 用于训练模型
- validation set 用于模型选择和调参
- test set 用于评估最终模型的泛化能力

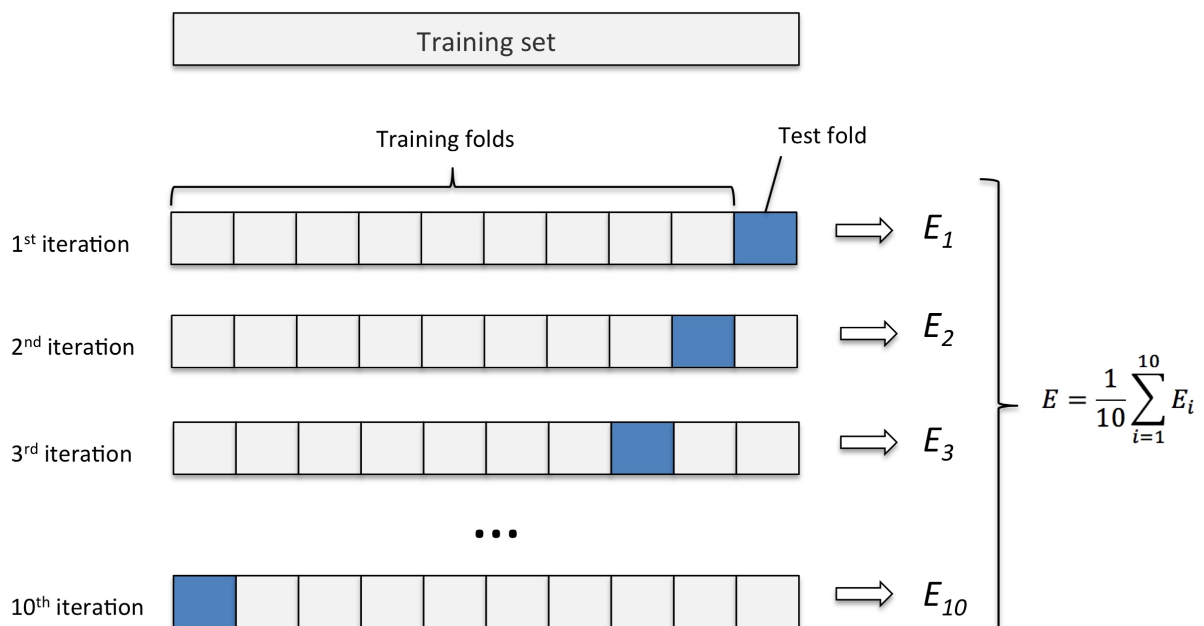
[back to top]



K-fold cross validation

重复 hold out method k 次. 保留 test set，剩下数据随机分为 k 组, 将其中一组留作 validation set, 其余做 training data, 更换 validation 组重复训练 k 次

[back to top]



```
import numpy as np
from sklearn.cross_validation import KFold

kfold = KFold(n=len(X_train), n_folds=10, random_state=1)

scores = []
for k, (train, test) in enumerate(kfold):
    pipe_lr.fit(X_train[train], y_train[train])
    score = pipe_lr.score(X_train[test], y_train[test])
    scores.append(score)
    print('Fold: %s, Class dist.: %s, Acc: %.3f' % (k+1, np.binc
ount(y_train[train]), score))

print('\nCV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(
scores)))
```

```
Fold: 1, Class dist.: [256 153], Acc: 0.891
Fold: 2, Class dist.: [254 155], Acc: 0.957
Fold: 3, Class dist.: [258 151], Acc: 0.978
Fold: 4, Class dist.: [257 152], Acc: 0.913
Fold: 5, Class dist.: [255 154], Acc: 0.935
Fold: 6, Class dist.: [258 152], Acc: 0.978
Fold: 7, Class dist.: [257 153], Acc: 0.933
Fold: 8, Class dist.: [254 156], Acc: 0.956
Fold: 9, Class dist.: [259 151], Acc: 0.978
Fold: 10, Class dist.: [257 153], Acc: 0.956

CV accuracy: 0.947 +/- 0.028
```

Stratified k-fold cross validation

Stratified k-fold CV 方法在切分数据时，会尽量保持各标签的比例，从而获得更准确的模型效果评估

[back to top]

```
from sklearn.cross_validation import StratifiedKFold

kfold = StratifiedKFold(y=y_train,
                        n_folds=10,
                        random_state=1)


scores = []
for k, (train, test) in enumerate(kfold):
    pipe_lr.fit(X_train[train], y_train[train])
    score = pipe_lr.score(X_train[test], y_train[test])
    scores.append(score)
    print('Fold: %s, Class dist.: %s, Acc: %.3f' % (k+1, np.binc
ount(y_train[train]), score))

print('\nCV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(
scores)))
```

```
Fold: 1, Class dist.: [256 153], Acc: 0.891
Fold: 2, Class dist.: [256 153], Acc: 0.978
Fold: 3, Class dist.: [256 153], Acc: 0.978
Fold: 4, Class dist.: [256 153], Acc: 0.913
Fold: 5, Class dist.: [256 153], Acc: 0.935
Fold: 6, Class dist.: [257 153], Acc: 0.978
Fold: 7, Class dist.: [257 153], Acc: 0.933
Fold: 8, Class dist.: [257 153], Acc: 0.956
Fold: 9, Class dist.: [257 153], Acc: 0.978
Fold: 10, Class dist.: [257 153], Acc: 0.956

CV accuracy: 0.950 +/- 0.029
```

sklearn 里 cross_val_score 函数默认使用 stratified k-fold CV

```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(estimator=pipe_lr,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=-1)  # 使用 CPUs 的内核数
print('CV accuracy scores:\n %s' % scores)
print('CV accuracy:\n %.3f +/- %.3f' % (np.mean(scores), np.std(
scores)))
```

```
CV accuracy scores:
 [ 0.89130435  0.97826087  0.97826087  0.91304348  0.93478261  0
.97777778
  0.93333333  0.95555556  0.97777778  0.95555556]
CV accuracy:
 0.950 +/- 0.029
```

Learning and validation curves

diagnose if a learning algorithm has a problem with overfitting (high variance) or underfitting (high bias)

[back to top]

Diagnosing bias and variance problems with learning curves

plotting the training and test accuracies as functions of the sample size

[back to top]

```
# 画 learning curve, Accuracy 与 training sample size的关系
```

```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.learning_curve import learning_curve

pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', LogisticRegression(penalty='l2', C=0
.1, random_state=0))])

# learning_curve 中的 scores 通过 stratified k-fold CV 获得
train_sizes, train_scores, test_scores =\
    learning_curve(estimator=pipe_lr, X=X_train, y=y_train,
                   train_sizes=np.linspace(0.1, 1.0, 10),  # 调整
训练集中样本的数量
                   cv=10, n_jobs=-1)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

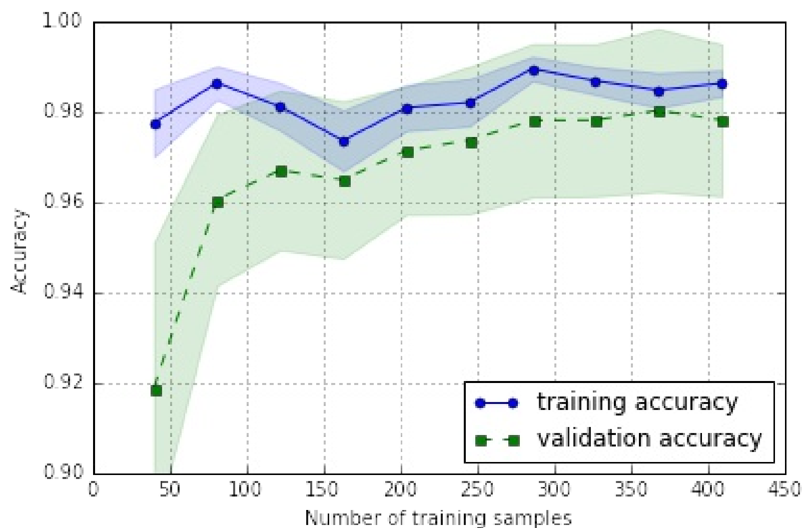
# plot train_mean
plt.plot(train_sizes, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')

# plot train_std
plt.fill_between(train_sizes,
                 train_mean + train_std,
                 train_mean - train_std,
                 alpha=0.15, color='blue')
# plot test_mean
plt.plot(train_sizes, test_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')

# plot test_std
plt.fill_between(train_sizes,
                 test_mean + test_std,
                 test_mean - test_std,
```

```
                   alpha=0.15, color='green')

plt.grid()
plt.xlabel('Number of training samples')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim([0.9, 1.0])
plt.tight_layout()
# plt.savefig('./figures/learning_curve.png', dpi=300)
```



Addressing overfitting and underfitting with validation curves

plotting the training and test accuracies as functions of the model parameters

[back to top]

```
# validation curve, useful tool for improving the performance of
 a model
from sklearn.learning_curve import validation_curve

# 设定参数选项
param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
```

```
# 通过 CV 获得不同参数的模型效果
train_scores, test_scores = \
    validation_curve(estimator=pipe_lr,
                     X=X_train, y=y_train,
                     param_name='clf__C',  # 用 pipe_lr.get_para
ms() 找到参数对应的名称
                     param_range=param_range, cv=10)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

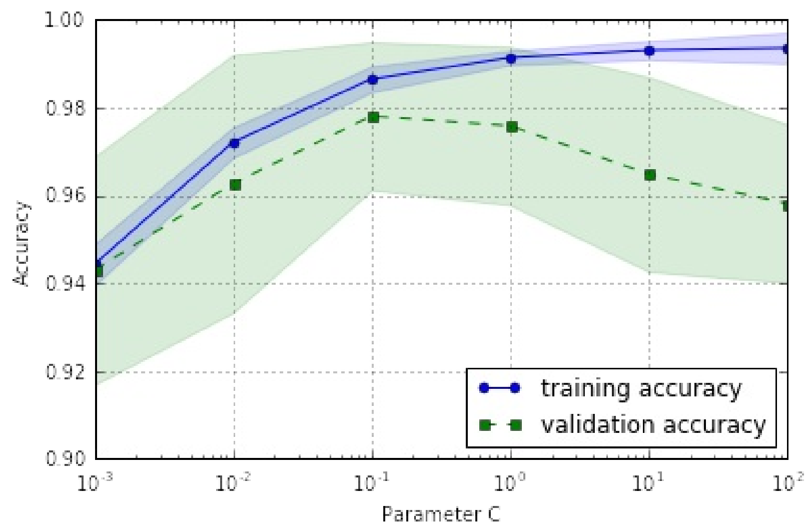
plt.plot(param_range, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')

plt.fill_between(param_range, train_mean + train_std,
                 train_mean - train_std, alpha=0.15,
                 color='blue')

plt.plot(param_range, test_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')

plt.fill_between(param_range,
                 test_mean + test_std,
                 test_mean - test_std,
                 alpha=0.15, color='green')

plt.grid()
plt.xscale('log')
plt.legend(loc='lower right')
plt.xlabel('Parameter C')
plt.ylabel('Accuracy')
plt.ylim([0.9, 1.0])
plt.tight_layout()
# plt.savefig('./figures/validation_curve.png', dpi=300)
```



从图中可以看出，随着 C 增加 (regularization 减小)，模型由 underfit -> optimal -> overfit
最佳 C 参数值应选用 0.1

Grid search

[back to top]

Tuning hyperparameters via grid search

finding the optimal combination of hyperparameter values.

[back to top]

```
# brute-force exhaustive search, 遍历
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC

svc = SVC(random_state=1)

param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0
]
param_grid = {'C': param_range}

gs = GridSearchCV(estimator=svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)  # use all CPU
gs = gs.fit(X_train, y_train)
print(gs.best_score_)  # validation accuracy best
print(gs.best_params_)
```

```
0.626373626374
{'C': 0.0001}
```

结合 pipeline 和 grid search

```
pipe_svc = Pipeline([('scl', StandardScaler()),
                     ('clf', SVC(random_state=1))])

# linear SVM: inverse regularization parameter C
# RBF kernel SVM: both C and gamma parameter
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0
]
param_grid = [{'clf__C': param_range,
               'clf__kernel': ['linear']},
              {'clf__C': param_range, 'clf__gamma': param_range,

               'clf__kernel': ['rbf']}]

gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)  # use all CPU
gs = gs.fit(X_train, y_train)
print(gs.best_score_)  # validation accuracy best
print(gs.best_params_)
```

```
0.978021978022
{'clf__C': 0.1, 'clf__kernel': 'linear'}
```

```
# 看在测试集上的效果
clf = gs.best_estimator_
clf.fit(X_train, y_train)
print('Test accuracy: %.3f' % clf.score(X_test, y_test))
```

```
Test accuracy: 0.965
```

Randomized search

Although grid search is a powerful approach for finding the optimal set of parameters, the evaluation of all possible parameter combinations is also computationally very expensive.
An alternative approach to sampling different parameter combinations using scikit-learn is randomized search.

[back to top]

```
from scipy.stats import expon
from sklearn.grid_search import RandomizedSearchCV

pipe_svc = Pipeline([('scl', StandardScaler()),
                     ('clf', SVC(random_state=1))])


param_dist = {'clf__C': expon(scale=100),
              'clf__gamma': expon(scale=0.1),
              'clf__kernel': ['rbf']}

np.random.seed(0)
rs = RandomizedSearchCV(estimator=pipe_svc,
                        param_distributions=param_dist,
                        n_iter=20, scoring='accuracy',
                        cv=10, n_jobs=-1)

rs = rs.fit(X_train, y_train)
print(rs.best_score_)
print(rs.best_params_)
```

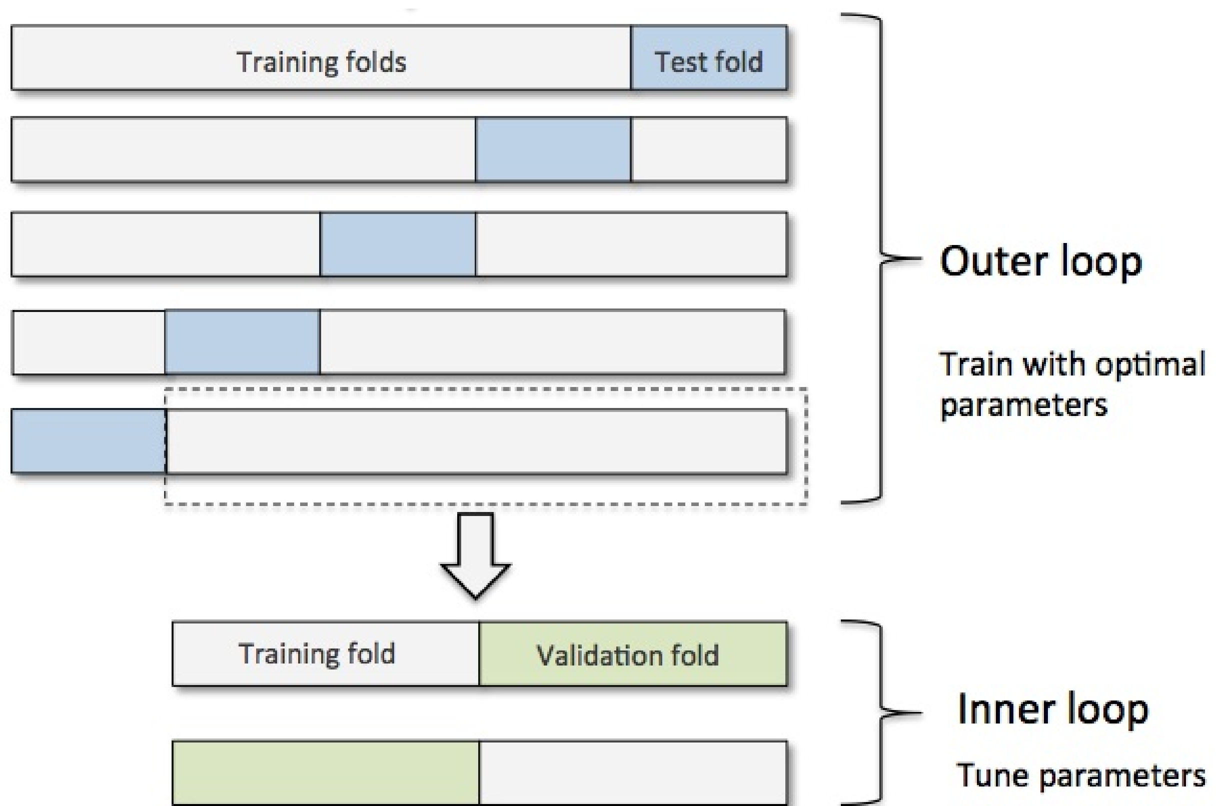
```
0.975824175824
{'clf__gamma': 0.009116102911900048, 'clf__C': 7.368535491284788
, 'clf__kernel': 'rbf'}
```

```
clf = rs.best_estimator_
clf.fit(X_train, y_train)
print('Test accuracy: %.3f' % clf.score(X_test, y_test))
```

```
Test accuracy: 0.974
```

Model selection with nested cross-validation

[back to top]



用 nested cross validation 来比较 SVM 和 decision tree 模型

```
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=2)

scores = cross_val_score(gs, X_train, y_train, scoring='accuracy'
, cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(sc
ores)))
```

```
CV accuracy: 0.965 +/- 0.025
```

```
from sklearn.tree import DecisionTreeClassifier
gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0
),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7
, None]}],
                  scoring='accuracy',
                  cv=2)
scores = cross_val_score(gs, X_train, y_train, scoring='accuracy'
, cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(sc
ores)))
```

```
CV accuracy: 0.921 +/- 0.029
```

从结果来看，应该选用 SVM 模型

练习1：自己来写一个函数，将数据分成两个部分

练习2：使用信贷数据集，尝试参数调优

(#sections)

第6章：非监督学习方法

Sections

Clustering is the task of gathering samples into groups of similar samples according to some predefined similarity or distance (dissimilarity) measure, such as the Euclidean distance.

Here are some common applications of clustering algorithms:

- Compression for data reduction
- Summarizing data as a reprocessing step for recommender systems
- Similarly:
    - grouping related web news (e.g. Google News) and web search results
    - grouping related stock quotes for investment portfolio management
    - building customer profiles for market analysis
- Building a code book of prototype samples for unsupervised feature extraction

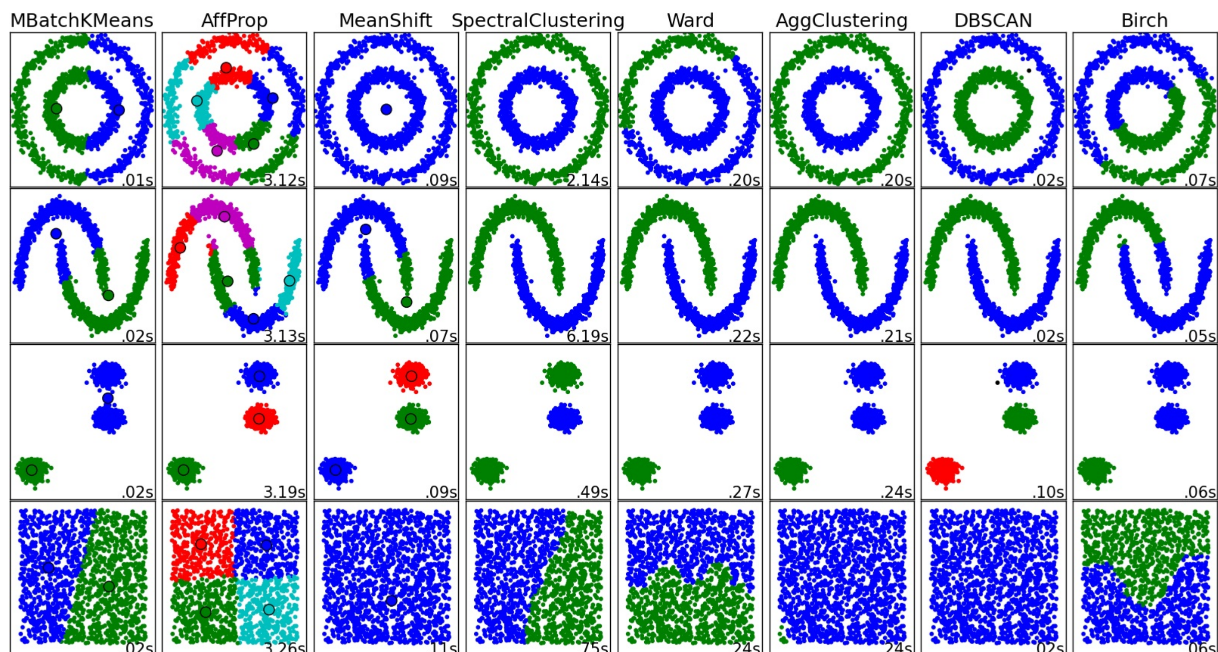
Some notable clustering routines

[back to top]

The following are two well-known clustering algorithms.

- `sklearn.cluster.KMeans` :
  The simplest, yet effective clustering algorithm. Needs to be provided with the number of clusters in advance, and assumes that the data is normalized as input (but use a PCA model as preprocessor).
- `sklearn.cluster.MeanShift` :
  Can find better looking clusters than KMeans but is not scalable to high number of samples.
- `sklearn.cluster.DBSCAN` :
  Can detect irregularly shaped clusters based on density, i.e. sparse regions in the input space are likely to become inter-cluster boundaries. Can also detect outliers (samples that are not part of a cluster).
- `sklearn.cluster.AffinityPropagation` :
  Clustering algorithm based on message passing between data points.
- `sklearn.cluster.SpectralClustering` :
  KMeans applied to a projection of the normalized graph Laplacian: finds normalized graph cuts if the affinity matrix is interpreted as an adjacency matrix of a graph.
- `sklearn.cluster.Ward` :
  Ward implements hierarchical clustering based on the Ward algorithm, a variance-minimizing approach. At each step, it minimizes the sum of squared differences within all clusters (inertia criterion).

Of these, Ward, SpectralClustering, DBSCAN and Affinity propagation can also work with precomputed similarity matrices.

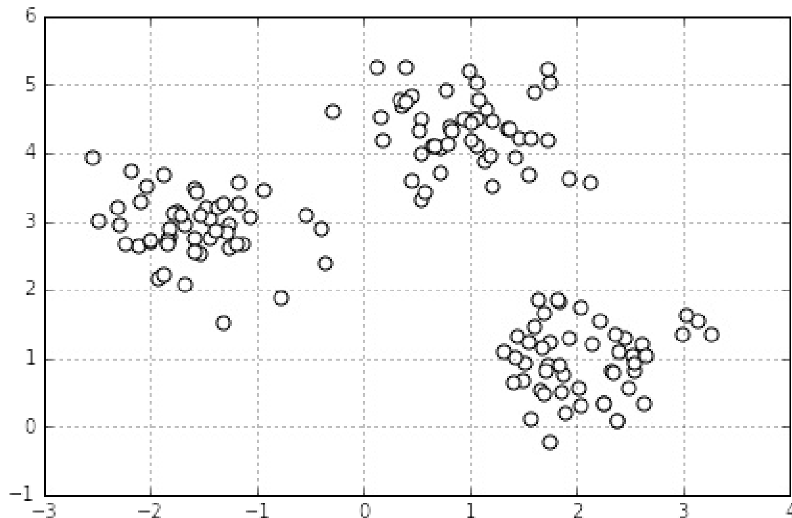


Grouping objects by similarity using k-means

[back to top]

```
# make dataset
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=150,
                  n_features=2,
                  centers=3, # 从三类中抽取
                  cluster_std=0.5,
                  shuffle=True,
                  random_state=0)
```

```
# 画个图看看
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(X[:,0], X[:,1],  c='white', marker='o', s=50)
plt.grid()
plt.tight_layout()
#plt.savefig('./figures/spheres.png', dpi=300)
```



k-means 算法

1. Randomly pick `k` centroids from the sample points as initial cluster centers.
2. Assign each sample to the nearest centroid $\mu^{(j)}, j \in \{1, \ldots, k\}$.
3. Move the centroids to the center of the samples that were assigned to it.
4. Repeat the steps 2 and 3 until the cluster assignment do not change or a user-defined tolerance or a maximum number of iterations is reached.

Visualizing K-Means Clustering

如何来测量两个物体之间的相似度, similarity
或者如何表示两个物体之间的距离, distance

最常见的一种距离度量是 squared Euclidean distance:
$d(x, y)^2 = \sum_{j=1}^{m} (x_j - y_j)^2 = \|x - y\|_2^2$

k-means 可转化为最优化的问题，最小化 within-cluster sum of squared errors (SSE)，又称为 cluster inertia $SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} w^{(i,j)} \|x^{(i)} - \mu^{(j)}\|_2^2$ 其中 $\mu^{(j)}$ 是第 j 个聚类的中心，如果样本 i 在聚类 j 中，$w^{(i,j)} = 1$，否则 $w^{(i,j)} = 0$

k-means in Sklearn

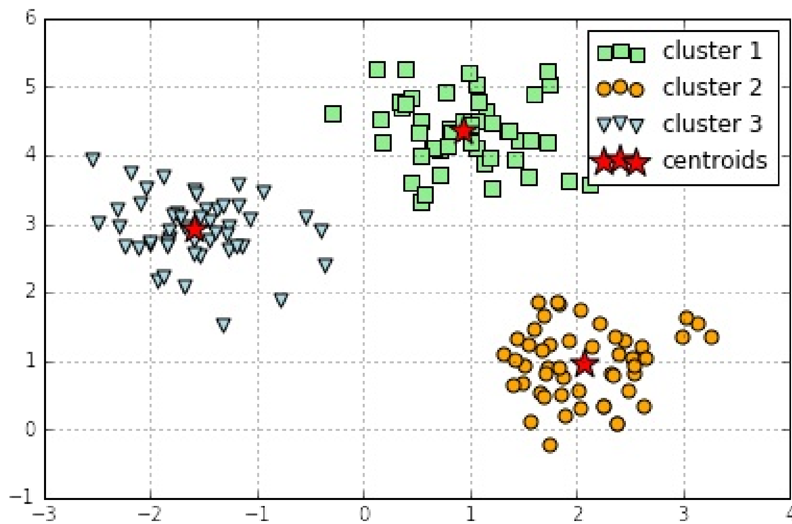
[back to top]

```
# 使用 sklearn 中 KMeans 来聚类
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3,  # k 是需要自己设定的，错误的 k 可能结果就不对

            init='random',
            n_init=10,  # 重复运行算法 10 次，选其中最好的聚类模型，避
免不好的初始化值带来的影响
            max_iter=300,
            tol=1e-04,
            random_state=0)

y_km = km.fit_predict(X)
```

```
plt.scatter(X[y_km==0,0],
            X[y_km==0,1],
            s=50,
            c='lightgreen',
            marker='s',
            label='cluster 1')
plt.scatter(X[y_km==1,0],
            X[y_km==1,1],
            s=50,
            c='orange',
            marker='o',
            label='cluster 2')
plt.scatter(X[y_km==2,0],
            X[y_km==2,1],
            s=50,
            c='lightblue',
            marker='v',
            label='cluster 3')
plt.scatter(km.cluster_centers_[:,0],
            km.cluster_centers_[:,1],
            s=250,
            marker='*',
            c='red',
            label='centroids')
plt.legend()
plt.grid()
plt.tight_layout()
#plt.savefig('./figures/centroids.png', dpi=300)
```



k-means++

[back to top]

k-means++ 算法通过改善 centroids 初始值的设置，来优化 k-means

1. Initialize an empty set M to store the k centroids being selected
2. Randomly choose the first centroid $\mu^{(j)}$ from the input samples and assign it to M
3. For each sample $x^{(i)}$ that is not in M, find the minimum squared distance $d(x^{(i)}, M)^2$ to any of the centroids in M
4. To randomly select the next centroid $\mu^{(p)}$, use a weighted probability distribution equal to $\frac{d(\mu^{(p)}, M)^2}{\sum_i d(x^{(i)}, M)^2}$
5. Repeat steps 2 and 3 until k centroids are chosen
6. Proceed with the classic k-means algorithm

sklearn 里使用 k-means++ 算法只需在 KMeans() 里设置 init="k-means++"

```
# 对比 k-means 和 k-means++
import numpy as np
from sklearn.utils import shuffle
from sklearn.utils import check_random_state
from sklearn.cluster import KMeans
```

```
random_state = np.random.RandomState(0)

# Number of run (with randomly generated dataset) for each strat
egy so as
# to be able to compute an estimate of the standard deviation
n_runs = 5

# k-means models can do several random inits so as to be able to
 trade
# CPU time for convergence robustness
n_init_range = np.array([1, 5, 10, 15, 20])

# Datasets generation parameters
n_samples_per_center = 100
grid_size = 3
scale = 0.1
n_clusters = grid_size ** 2


def make_data(random_state, n_samples_per_center, grid_size, sca
le):
    random_state = check_random_state(random_state)
    centers = np.array([[i, j]
                        for i in range(grid_size)
                        for j in range(grid_size)])
    n_clusters_true, n_features = centers.shape

    noise = random_state.normal(
        scale=scale, size=(n_samples_per_center, centers.shape[1
]))

    X = np.concatenate([c + noise for c in centers])
    y = np.concatenate([[i] * n_samples_per_center
                        for i in range(n_clusters_true)])
    return shuffle(X, y, random_state=random_state)


fig = plt.figure()
plots = []
```

```
legends = []

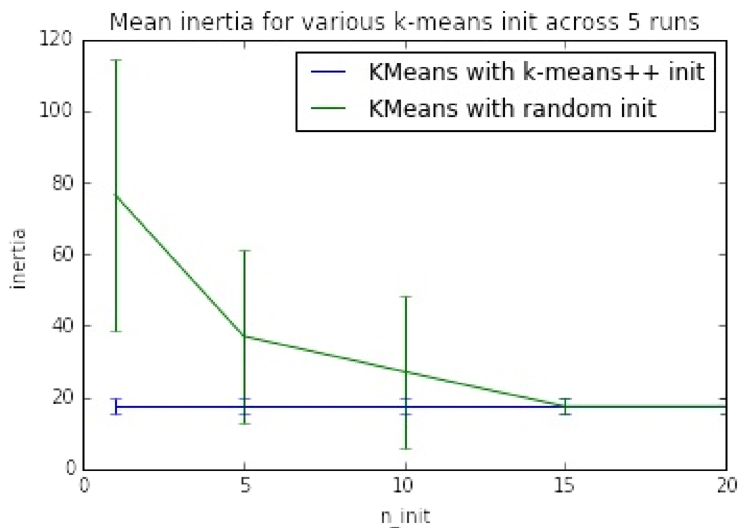
cases = [
    (KMeans, 'k-means++'),
    (KMeans, 'random')
]

for factory, init in cases:
    print("Evaluation of %s with %s init" % (factory.__name__, i
nit))
    inertia = np.empty((len(n_init_range), n_runs))

    for run_id in range(n_runs):
        X_data, y = make_data(run_id, n_samples_per_center, grid
_size, scale)
        for i, n_init in enumerate(n_init_range):
            kmean = factory(n_clusters=n_clusters, init=init, ra
ndom_state=run_id,
                             n_init=n_init).fit(X_data)
            inertia[i, run_id] = kmean.inertia_
    p = plt.errorbar(n_init_range, inertia.mean(axis=1), inertia
.std(axis=1))
    plots.append(p[0])
    legends.append("%s with %s init" % (factory.__name__, init))

plt.xlabel('n_init')
plt.ylabel('inertia')
plt.legend(plots, legends)
plt.title("Mean inertia for various k-means init across %d runs"
 % n_runs);
```

```
Evaluation of KMeans with k-means++ init
Evaluation of KMeans with random init
```



Implementing k-means in Python

[back to top]

```
import numpy as np
from sklearn.metrics import pairwise_distances


def get_initial_centroids(data, k, seed=None):
    '''Randomly choose k data points as initial centroids'''
    if seed is not None: # useful for obtaining consistent resul
ts
        np.random.seed(seed)
    n = data.shape[0] # number of data points

    # Pick K indices from range [0, N).
    rand_indices = np.random.randint(0, n, k)

    # Keep centroids as dense format, as many entries will be no
nzero due to averaging.
    # As long as at least one document in a cluster contains a w
ord,
    # it will carry a nonzero weight in the TF-IDF vector of the
 centroid.
    centroids = data[rand_indices,:]
```

```
    return centroids


def smart_initialize(data, k, seed=None):
    '''Use k-means++ to initialize a good set of centroids'''
    if seed is not None: # useful for obtaining consistent results
        np.random.seed(seed)
    centroids = np.zeros((k, data.shape[1]))

    # Randomly choose the first centroid.
    # Since we have no prior knowledge, choose uniformly at random
    idx = np.random.randint(data.shape[0])
    centroids[0] = data[idx,:]
    # Compute distances from the first centroid chosen to all the other data points
    distances = pairwise_distances(data, centroids[0:1], metric='euclidean').flatten()

    for i in xrange(1, k):
        # Choose the next centroid randomly, so that the probability for each data point to be chosen
        # is directly proportional to its squared distance from the nearest centroid.
        # Roughtly speaking, a new centroid should be as far as from ohter centroids as possible.
        idx = np.random.choice(data.shape[0], 1, p=distances/sum(distances))
        centroids[i] = data[idx,:]
        # Now compute distances from the centroids to all data points
        distances = np.min(pairwise_distances(data, centroids[0:i+1], metric='euclidean'),axis=1)

    return centroids


def assign_clusters(data, centroids):
```

```
    # Compute distances between each data point and the set of c
entroids:
    distances_from_centroids = pairwise_distances(data, centroid
s)

    # Compute cluster assignments for each data point:
    cluster_assignment = np.argmin(distances_from_centroids, axi
s=1)

    return cluster_assignment


def revise_centroids(data, k, cluster_assignment):
    new_centroids = []
    for i in xrange(k):
        # Select all data points that belong to cluster i. Fill
in the blank (RHS only)
        member_data_points = data[cluster_assignment == i]
        # Compute the mean of the data points. Fill in the blank
 (RHS only)
        centroid = member_data_points.mean(axis=0)
        new_centroids.append(centroid)
    new_centroids = np.array(new_centroids)
    return new_centroids


def kmeans(data, k, init='kmeans++', maxiter=100, seed=None):
    # Initialize centroids
    if init == 'kmeans++':
        centroids = smart_initialize(data, k, seed)
    else:
        centroids = get_initial_centroids(data, k, seed)
    prev_cluster_assignment = None

    for itr in xrange(maxiter):
        # 1. Make cluster assignments using nearest centroids
        cluster_assignment = assign_clusters(data, centroids)

        # 2. Compute a new centroid for each of the k clusters,
averaging all data points assigned to that cluster.
```

```
        centroids = revise_centroids(data, k, cluster_assignment
)

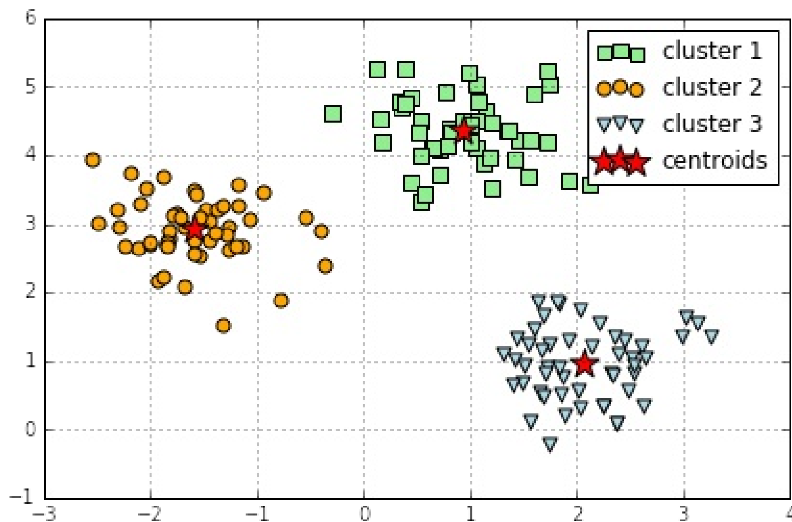
        # Check for convergence: if none of the assignments chan
ged, stop
        if prev_cluster_assignment is not None and \
          (prev_cluster_assignment==cluster_assignment).all():
            break

        prev_cluster_assignment = cluster_assignment[:]

    return centroids, cluster_assignment
```

```
centers, y_km = kmeans(X, 3, seed=0)
```

```
plt.scatter(X[y_km==0,0],
            X[y_km==0,1],
            s=50,
            c='lightgreen',
            marker='s',
            label='cluster 1')
plt.scatter(X[y_km==1,0],
            X[y_km==1,1],
            s=50,
            c='orange',
            marker='o',
            label='cluster 2')
plt.scatter(X[y_km==2,0],
            X[y_km==2,1],
            s=50,
            c='lightblue',
            marker='v',
            label='cluster 3')
plt.scatter(centers[:,0],
            centers[:,1],
            s=250,
            marker='*',
            c='red',
            label='centroids')
plt.legend()
plt.grid()
plt.tight_layout()
#plt.savefig('./figures/centroids.png', dpi=300)
```



Using the elbow method to find the optimal number of clusters

通常我们并不知道数据能分成几个聚类，所以能有办法选择合适 k 值非常重要

[back to top]

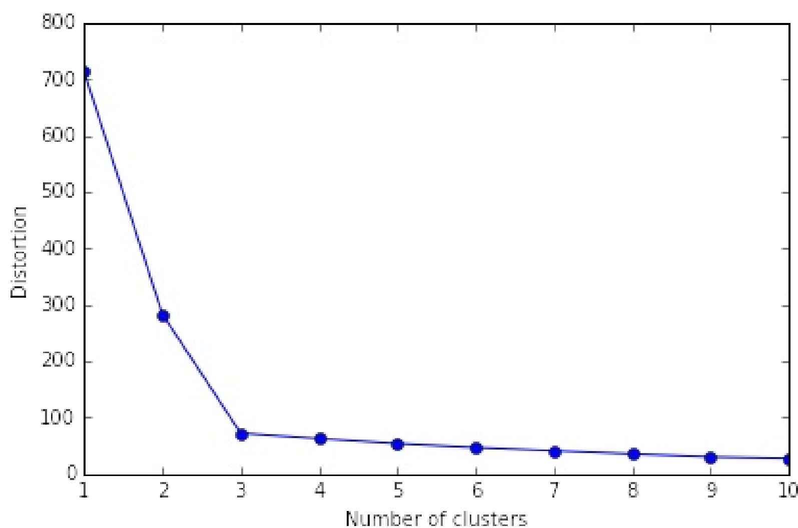
判断聚类效果可用 within-cluster SSE (Distortion)，这个可由 KMeans() 中的 inertia_ 属性获得

```
print('Distortion: %.2f' % km.inertia_)
```

```
Distortion: 72.48
```

The Elbow method is a "rule-of-thumb" approach to finding the optimal number of clusters.

```
# elbow method 方法就是需要找出当 distortion 变化非常快时的 k 值，也即找拐点
distortions = []
for i in range(1, 11):
    km = KMeans(n_clusters=i,
                init='k-means++',
                n_init=10,
                max_iter=300,
                random_state=0)
    km.fit(X)
    distortions.append(km.inertia_)
plt.plot(range(1,11), distortions , marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()
#plt.savefig('./figures/elbow.png', dpi=300)
```



从图中就可以看出, 3 是拐点, 所以 k=3 是最好的选择

Quantifying the quality of clustering via silhouette plots

[back to top]

另一种评价聚类效果的方法是 silhouette analysis，衡量的是一个类别中的样本是否足够紧凑组合

计算 $x^{(i)}$ 的 silhouette coefficient 的步骤为：

1. Calculate the cluster cohesion $a^{(i)}$ as the average distance between a sample $x^{(i)}$ and all other points in the same cluster.
2. Calculate the cluster separation $b^{(i)}$ from the next closest cluster as the average distance between the sample $x^{(i)}$ and all samples in the nearest cluster.
3. Calculate the silhouette $s^{(i)}$ as the difference between cluster cohesion and separation divided by the greater of the two, as shown here:
   $s^{(i)} = \frac{b^{(i)} - a^{(i)}}{max\left\{b^{(i)}, a^{(i)}\right\}}$

```
import numpy as np
from matplotlib import cm
from sklearn.metrics import silhouette_samples  # silhouette coe
fficient

km = KMeans(n_clusters=3,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km = km.fit_predict(X)

cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(X, y_km, metric='euclidean'
)
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
```

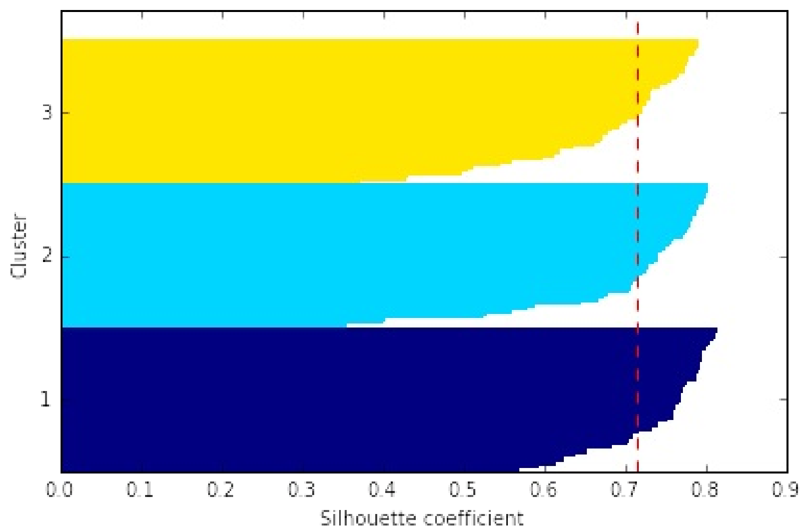
```
    color = cm.jet(i / float(n_clusters))
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, h
eight=1.0,
            edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
# plt.savefig('./figures/silhouette.png', dpi=300)
```



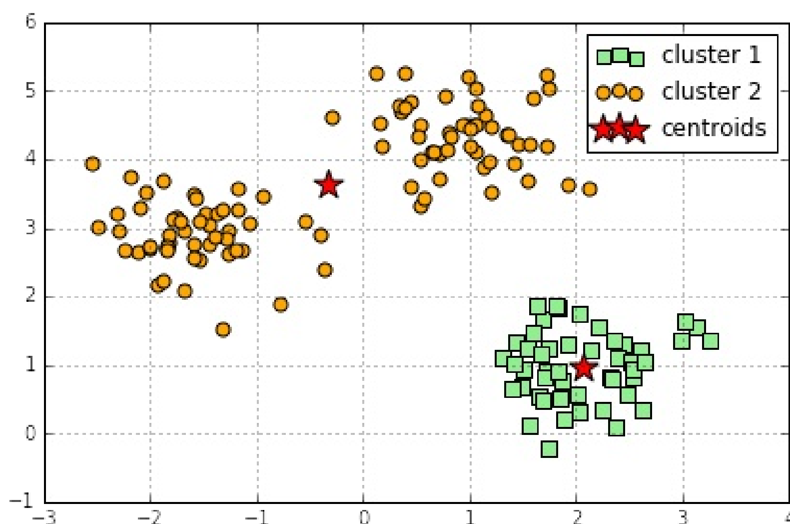
红线表示所有数据 silhouette coef 的平均值，它可作为聚类模型的一个度量指标

Comparison to "bad" clustering:

```
km = KMeans(n_clusters=2, # 设定为2
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km = km.fit_predict(X)

plt.scatter(X[y_km==0,0],
            X[y_km==0,1],
            s=50,
            c='lightgreen',
            marker='s',
            label='cluster 1')
plt.scatter(X[y_km==1,0],
            X[y_km==1,1],
            s=50,
            c='orange',
            marker='o',
            label='cluster 2')

plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1],
s=250, marker='*', c='red', label='centroids')
plt.legend()
plt.grid()
plt.tight_layout()
#plt.savefig('./figures/centroids_bad.png', dpi=300)
```



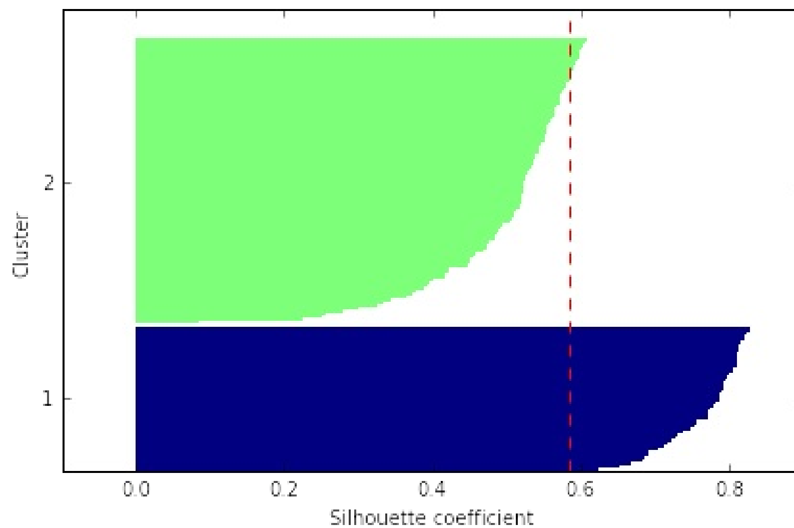
```
# evaluate the results
cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(X, y_km, metric='euclidean'
)
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(i / float(n_clusters))
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, h
eight=1.0,
            edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
# plt.savefig('./figures/silhouette_bad.png', dpi=300)
```



Organizing clusters as a hierarchical tree

[back to top]

One nice feature of hierachical clustering is that we can visualize the results as a dendrogram, a hierachical tree. Using the visualization, we can then decide how "deep" we want to cluster the dataset by setting a "depth" threshold. Or in other words, we don't need to make a decision about the number of clusters upfront.

Agglomerative and divisive hierarchical clustering

Furthermore, we can distinguish between 2 main approaches to hierarchical clustering: Divisive clustering and agglomerative clustering. In agglomerative clustering, we start with a single sample from our dataset and iteratively merge it with other samples to form clusters -- we can see it as a bottom-up approach for building the clustering dendrogram.
In divisive clustering, however, we start with the whole dataset as one cluster, and we iteratively split it into smaller subclusters -- a top-down approach.

In this notebook, we will use **agglomerative** clustering.

Single and complete linkage

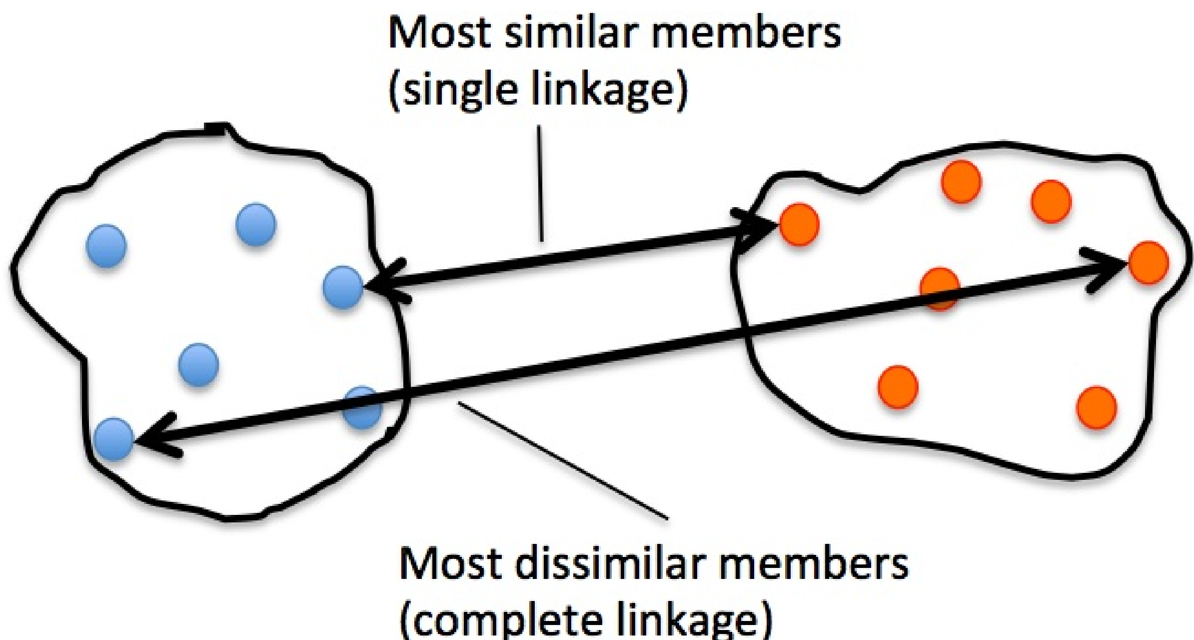
Now, the next question is how we measure the similarity between samples. One approach is the familiar Euclidean distance metric that we already used via the K-Means algorithm. as a refresher, the distance between 2 m-dimensional vectors $\mathbf{p}$ and $\mathbf{q}$ can be computed as:

$$\begin{align} \mathrm{d}(\mathbf{q},\mathbf{p}) & = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_m-p_m)^2} \\[8pt] & = \sqrt{\sum_{j=1}^m (q_j-p_j)^2}.\end{align}$$

However, that's the distance between 2 samples. Now, how do we compute the similarity between subclusters of samples in order to decide which clusters to merge when constructing the dendrogram? I.e., our goal is to iteratively merge the most similar pairs of clusters until only one big cluster remains. There are many different approaches to this, for example single and complete linkage.

In single linkage, we take the pair of the most similar samples (based on the Euclidean distance, for example) in each cluster, and merge the two clusters which have the most similar 2 members into one new, bigger cluster.

In complete linkage, we compare the pairs of the two most dissimilar members of each cluster with each other, and we merge the 2 clusters where the distance between its 2 most dissimilar members is smallest.



```
# 生成数据
import pandas as pd
import numpy as np

np.random.seed(123)

variables = ['X', 'Y', 'Z']
labels = ['ID_0','ID_1','ID_2','ID_3','ID_4']

X = np.random.random_sample([5,3])*10
df = pd.DataFrame(X, columns=variables, index=labels)
df
```

	X	Y	Z
ID_0	6.964692	2.861393	2.268515
ID_1	5.513148	7.194690	4.231065
ID_2	9.807642	6.848297	4.809319
ID_3	3.921175	3.431780	7.290497
ID_4	4.385722	0.596779	3.980443

Performing hierarchical clustering on a distance matrix

[back to top]

```
# calculate the distance matrix as input for the hierarchical cl
ustering algorithm
from scipy.spatial.distance import pdist,squareform

row_dist = pd.DataFrame(squareform(pdist(df, metric='euclidean')
), columns=labels, index=labels)
row_dist
```

	ID_0	ID_1	ID_2	ID_3	ID_4
ID_0	0.000000	4.973534	5.516653	5.899885	3.835396
ID_1	4.973534	0.000000	4.347073	5.104311	6.698233
ID_2	5.516653	4.347073	0.000000	7.244262	8.316594
ID_3	5.899885	5.104311	7.244262	0.000000	4.382864
ID_4	3.835396	6.698233	8.316594	4.382864	0.000000

We can either pass a condensed distance matrix (upper triangular) from the pdist function, or we can pass the "original" data array and define the metric='euclidean' argument in linkage. However, we should not pass the squareform distance matrix, which would yield different distance values although the overall clustering could be the same.

```
# 1. incorrect approach: Squareform distance matrix

from scipy.cluster.hierarchy import linkage

row_clusters = linkage(row_dist, method='complete', metric='eucl
idean')
pd.DataFrame(row_clusters,
             columns=['row label 1', 'row label 2', 'distance',
'no. of items in clust.'],
             index=['cluster %d' %(i+1) for i in range(row_clust
ers.shape[0])])
```

	row label 1	row label 2	distance	no. of items in clust.
cluster 1	0.0	4.0	6.521973	2.0
cluster 2	1.0	2.0	6.729603	2.0
cluster 3	3.0	5.0	8.539247	3.0
cluster 4	6.0	7.0	12.444824	5.0

```
# 2. correct approach: Condensed distance matrix

row_clusters = linkage(pdist(df, metric='euclidean'), method='co
mplete')
pd.DataFrame(row_clusters,
             columns=['row label 1', 'row label 2', 'distance',
'no. of items in clust.'],
             index=['cluster %d' %(i+1) for i in range(row_clust
ers.shape[0])])
```

	row label 1	row label 2	distance	no. of items in clust.
cluster 1	0.0	4.0	3.835396	2.0
cluster 2	1.0	2.0	4.347073	2.0
cluster 3	3.0	5.0	5.899885	3.0
cluster 4	6.0	7.0	8.316594	5.0

```
# 3. correct approach: Input sample matrix

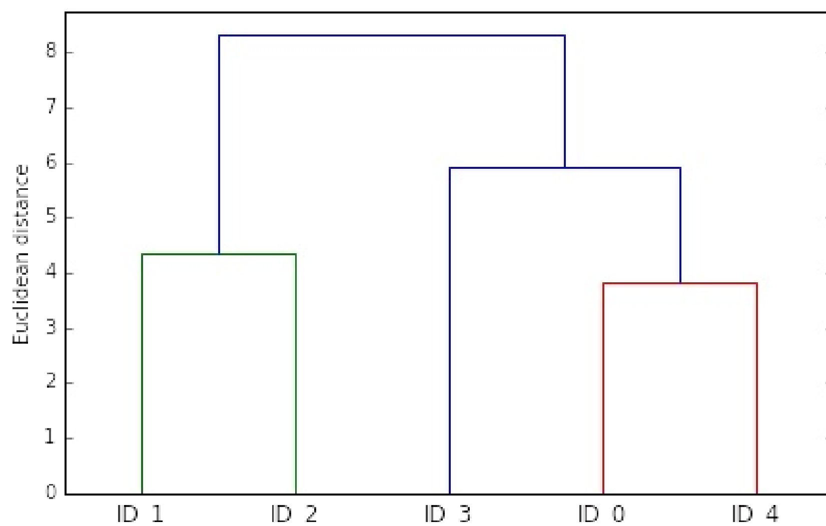
row_clusters = linkage(df.values, method='complete', metric='euc
lidean')
pd.DataFrame(row_clusters,
             columns=['row label 1', 'row label 2', 'distance',
'no. of items in clust.'],
             index=['cluster %d' %(i+1) for i in range(row_clust
ers.shape[0])])
```

	row label 1	row label 2	distance	no. of items in clust.
cluster 1	0.0	4.0	3.835396	2.0
cluster 2	1.0	2.0	4.347073	2.0
cluster 3	3.0	5.0	5.899885	3.0
cluster 4	6.0	7.0	8.316594	5.0

```
# 可视化结果, 使用 dendrogram
from scipy.cluster.hierarchy import dendrogram

# make dendrogram black (part 1/2)
# from scipy.cluster.hierarchy import set_link_color_palette
# set_link_color_palette(['black'])

row_dendr = dendrogram(row_clusters,
                       labels=labels,
                       # make dendrogram black (part 2/2)
                       # color_threshold=np.inf
                       )
plt.tight_layout()
plt.ylabel('Euclidean distance');
#plt.savefig('./figures/dendrogram.png', dpi=300, bbox_inches='t
ight')
```



Attaching dendrograms to a heat map

[back to top]

```
# plot row dendrogram
fig = plt.figure(figsize=(8, 8), facecolor='white')
axd = fig.add_axes([0.09, 0.1, 0.2, 0.6])

# note: for matplotlib < v1.5.1, please use orientation='right'
row_dendr = dendrogram(row_clusters, orientation='left')

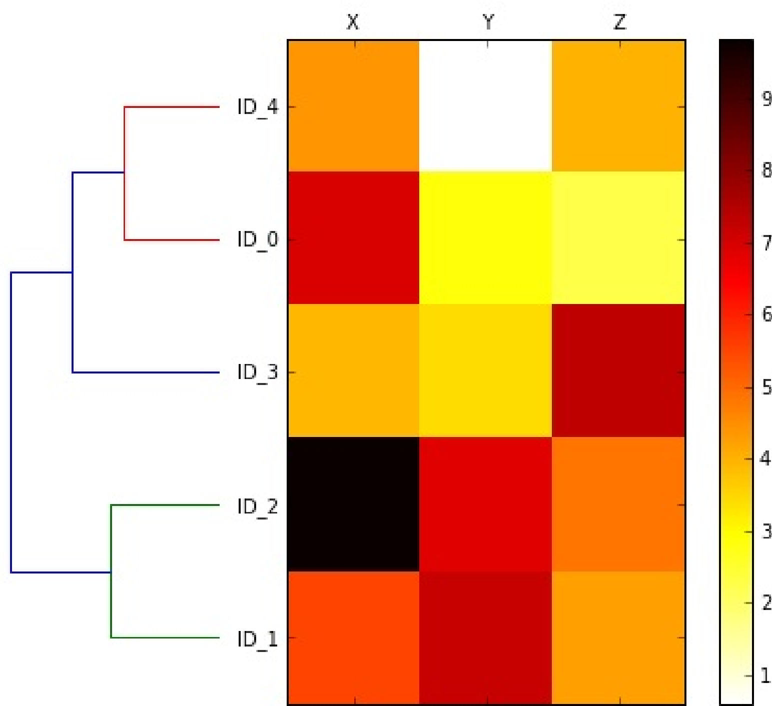
# reorder data with respect to clustering
df_rowclust = df.ix[row_dendr['leaves'][::-1]]

axd.set_xticks([])
axd.set_yticks([])

# remove axes spines from dendrogram
for i in axd.spines.values():
        i.set_visible(False)

# plot heatmap
axm = fig.add_axes([0.23, 0.1, 0.6, 0.6])  # x-pos, y-pos, width
, height
cax = axm.matshow(df_rowclust, interpolation='nearest', cmap='ho
t_r')
fig.colorbar(cax)
axm.set_xticklabels([''] + list(df_rowclust.columns))
axm.set_yticklabels([''] + list(df_rowclust.index));

# plt.savefig('./figures/heatmap.png', dpi=300)
```



Applying agglomerative clustering via scikit-learn

[back to top]

```
# 前面是用 scipy, 现在用 sklearn 来聚类
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=2, affinity='euclidean',
 linkage='complete')
labels = ac.fit_predict(X)
print('Cluster labels: %s' % labels)
```

```
Cluster labels: [0 1 1 0 0]
```

结果与前面一致

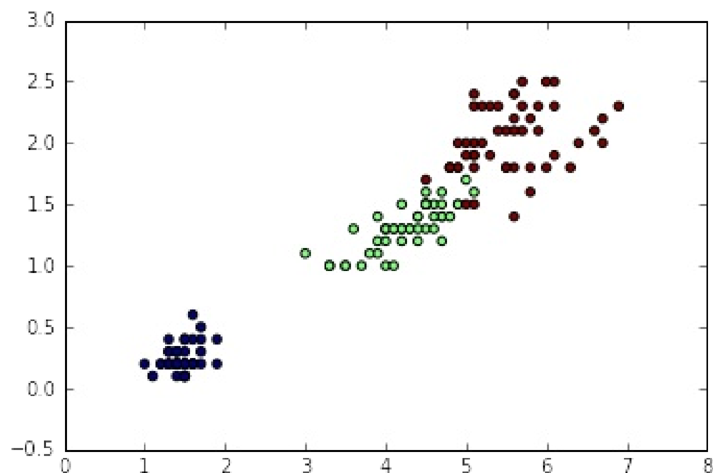
Applying agglomerative clustering with iris dataset

[back to top]

```
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
n_samples, n_features = X.shape

plt.scatter(X[:, 0], X[:, 1], c=y);
```

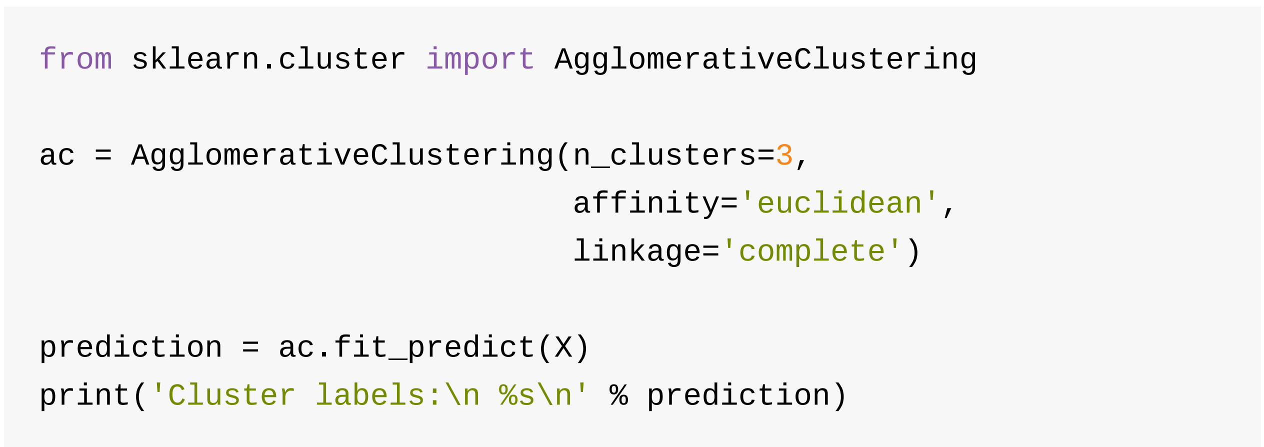



```
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram

clusters = linkage(X,
                   metric='euclidean',
                   method='complete')

dendr = dendrogram(clusters)

plt.ylabel('Euclidean Distance');
```



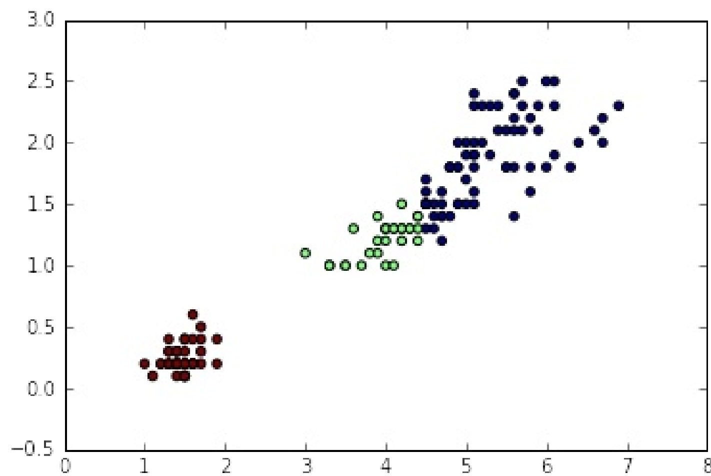

```
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=3,
                             affinity='euclidean',
                             linkage='complete')

prediction = ac.fit_predict(X)
print('Cluster labels:\n %s\n' % prediction)
```

```
Cluster labels:
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 1 0 0 0 1 0 1 1 1 1 0 1 1 0 1 0
 1 0 1 0 0
 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0
 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0
 0 0]
```

```
plt.scatter(X[:, 0], X[:, 1], c=prediction);
```



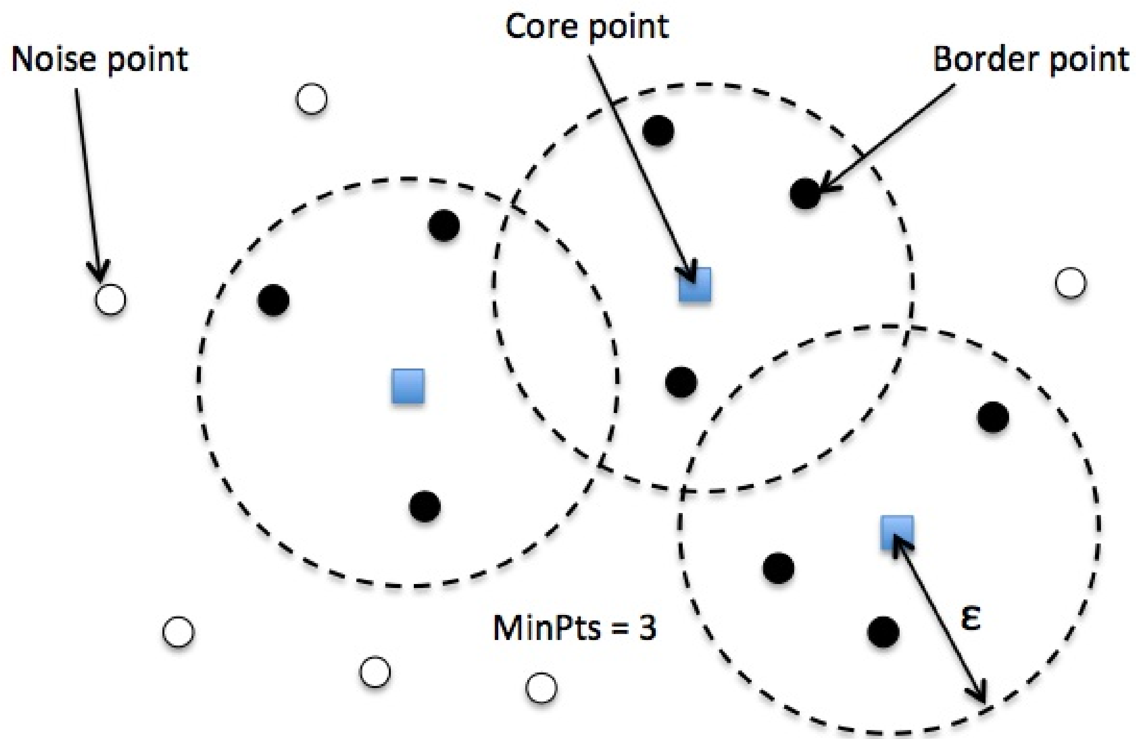
Locating regions of high density via DBSCAN

[back to top]

Another useful approach to clustering is *Density-based Spatial Clustering of Applications with Noise* (DBSCAN). In essence, we can think of DBSCAN as an algorithm that divides the dataset into subgroup based on dense regions of point.

In DBSCAN, we distinguish between 3 different "points":

- Core points: A core point is a point that has at least a minimum number of other points (MinPts) in its radius epsilon.
- Border points: A border point is a point that is not a core point, since it doesn't have enough MinPts in its neighborhood, but lies within the radius epsilon of a core point.
- Noise points: All other points that are neither core points nor border points.



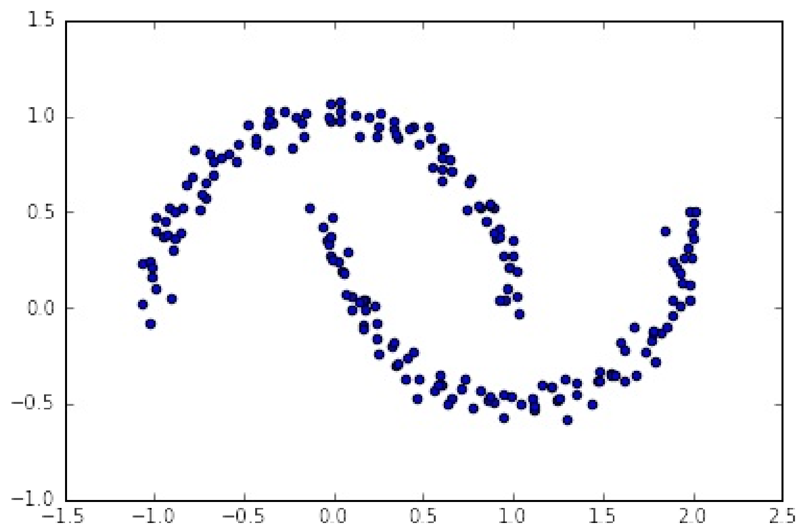
给每个点 label 之后, DBSCAN 算法就是下面两步:

1. Form a separate cluster for each core point or a connected group of core points (core points are connected if they are no farther away than ϵ).
2. Assign each border point to the cluster of its corresponding core point.

A nice feature about DBSCAN is that we don't have to specify a number of clusters upfront. However, it requires the setting of additional hyperparameters such as the value for MinPts and the radius epsilon.

```
# 生成半月形数据
# two visible, half-moon-shaped groups consisting of 100 sample
points each
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
plt.scatter(X[:,0], X[:,1])
plt.tight_layout()
#plt.savefig('./figures/moons.png', dpi=300)
```



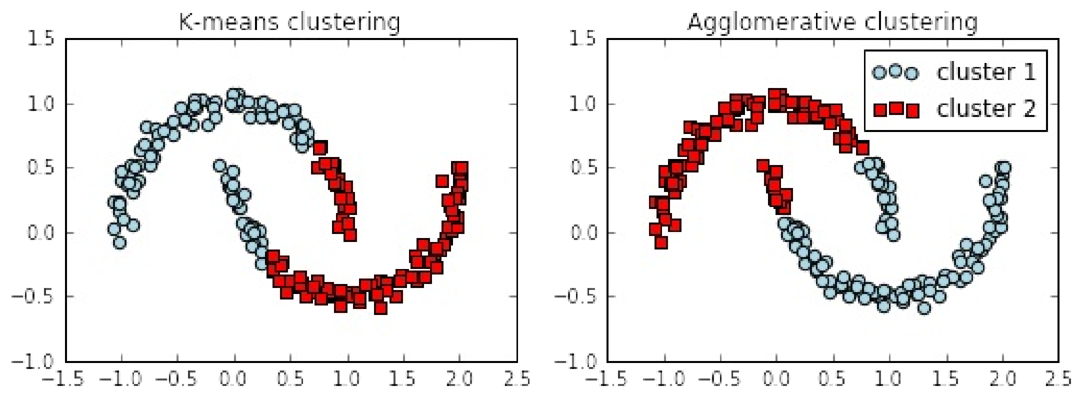
K-means and hierarchical clustering:

```
# complete linkage clustering
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(8,3))

km = KMeans(n_clusters=2, random_state=0)
y_km = km.fit_predict(X)
ax1.scatter(X[y_km==0,0], X[y_km==0,1], c='lightblue', marker='o'
, s=40, label='cluster 1')
ax1.scatter(X[y_km==1,0], X[y_km==1,1], c='red', marker='s', s=40
, label='cluster 2')
ax1.set_title('K-means clustering')

ac = AgglomerativeClustering(n_clusters=2, affinity='euclidean',
 linkage='complete')
y_ac = ac.fit_predict(X)
ax2.scatter(X[y_ac==0,0], X[y_ac==0,1], c='lightblue', marker='o'
, s=40, label='cluster 1')
ax2.scatter(X[y_ac==1,0], X[y_ac==1,1], c='red', marker='s', s=40
, label='cluster 2')
ax2.set_title('Agglomerative clustering')

plt.legend()
plt.tight_layout()
#plt.savefig('./figures/kmeans_and_ac.png', dpi=300)
```

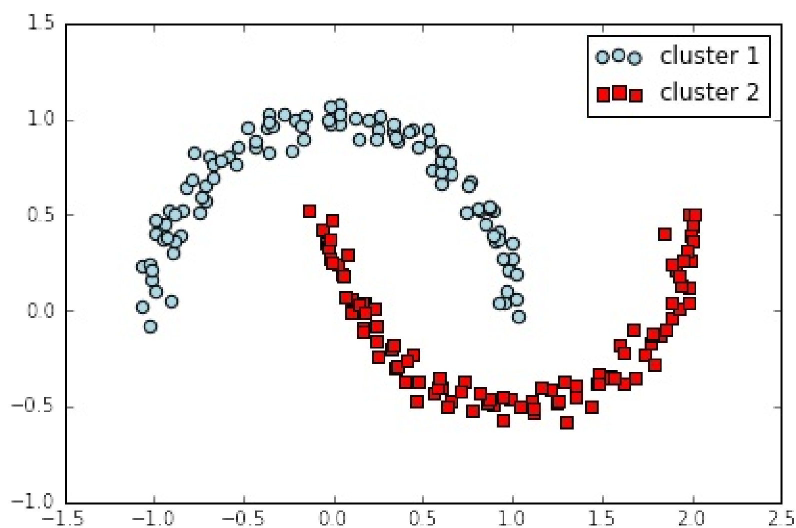


效果并不好，并不能完全 separated

Density-based clustering:

```
# DBSCAN
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.2, min_samples=5, metric='euclidean')
y_db = db.fit_predict(X)
plt.scatter(X[y_db==0,0], X[y_db==0,1], c='lightblue', marker='o'
, s=40, label='cluster 1')
plt.scatter(X[y_db==1,0], X[y_db==1,1], c='red', marker='s', s=40
, label='cluster 2')
plt.legend()
plt.tight_layout()
#plt.savefig('./figures/moons_dbscan.png', dpi=300)
```



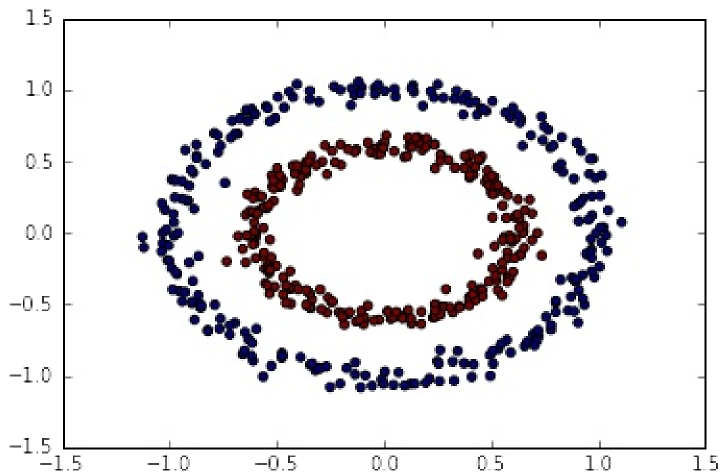
DBSCAN 能很好地对半月数据进行聚类

可以再试试圆环图形

```
from sklearn.datasets import make_circles

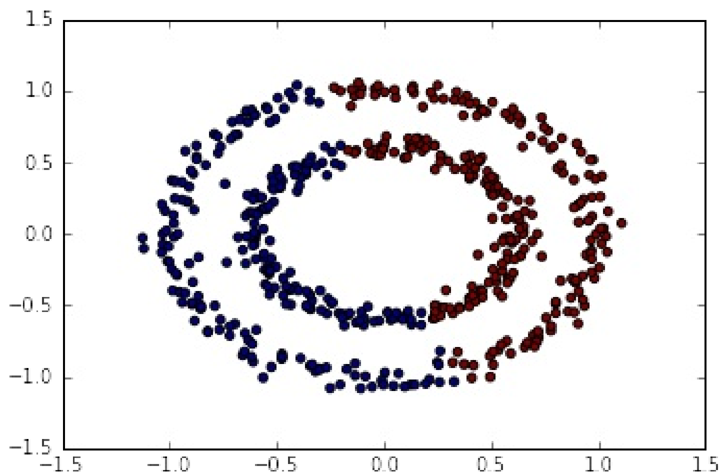
X, y = make_circles(n_samples=500,
                    factor=.6,
                    noise=.05)

plt.scatter(X[:, 0], X[:, 1], c=y);
```



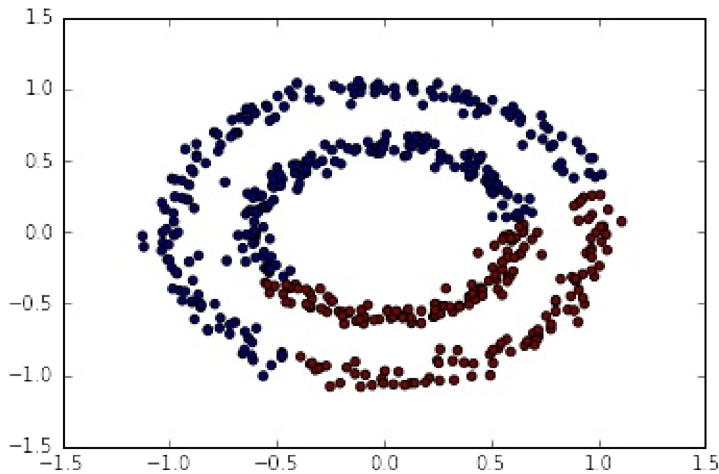
k-means

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=2)
predict = km.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=predict);
```



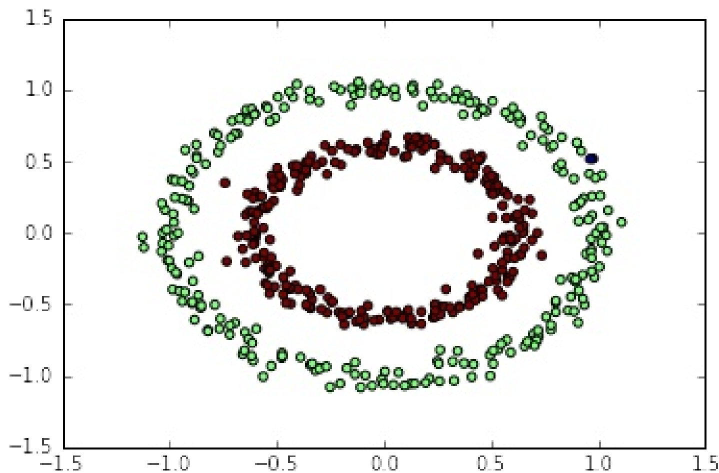
allomerative clustering

```
from sklearn.cluster import AgglomerativeClustering
ac = AgglomerativeClustering()
predict = ac.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=predict);
```



DBSCAN

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.15,
            min_samples=9,
            metric='euclidean')
predict = db.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=predict);
```



Learning from labeled and unlabeled data with label propagation

[back to top]

因为标注成本比较高，当你的训练数据集只有一部分数据是有标注的情况下，使用监督学习你只能扔掉那些没有标注的 X 。 而实际上，有标注的样本和无标注的样本之间是有关系的，这种关系信息也可以用来帮助学习。这就是半监督学习标签传播（Label Propagation）算法的思路。

它的基本逻辑是借助于近朱者赤的思路，也就是 KNN 的思路，如果 A 和 B 在 X 空间上很接近，那么 A 的 y 标签就可以传给 B。 进一步迭代下去，如果 C 和 B 也很接近，C 的标签也应该和 B 一样。 所以基本计算逻辑就是两步，第一步是计算样本间的距离，构建转移矩阵，第二步是将转移矩阵和 Y 矩阵相乘，Y 里面包括了已标注和未标注的两部分， 通过相乘可以将已标注的 Y 转播给未标注的 Y。 具体论文可以看这里。 在 Sklearn 模块中已经内置了这种算法，文档示例可以看这里。下面是用 Python 的 NumPy 模块实现的一个 toy demo。

```
import pandas as pd
import numpy as np
```

```
# 读入 iris 数据
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
n  = len(y)
```

```
# 切分数据
np.random.seed(42)
train_index = np.random.choice(n, int(0.6*n), replace = False)
test_index = np.setdiff1d(np.arange(n), train_index)
```

```
# 计算权重矩阵
sigma = X.var(axis = 0)
weights = np.zeros((n,n))

def weight_func(ind1, ind2, X=X, sigma=sigma):
    return np.exp(-np.sum((X[ind1] - X[ind2])**2/sigma))

for i in range(n):
    for j in range(n):
        weights[i,j] = weight_func(i,j)
```

```
##  标准化为转移矩阵
t = weights / weights.sum(axis=1)
```

```
# y转换形式
y_m = np.zeros((n, len(np.unique(y))))
for i in range(n):
    y_m[i,y[i]] = 1
```

```
## unlabel初始化, label记住
y_m[test_index] = np.random.random(y_m[test_index].shape)
clamp = y_m[train_index]
```

```
## 迭代计算
iter_n = 50
for _ in range(iter_n):
    y_m = t.dot(y_m)  # LP
    y_m = (y_m.T / y_m.sum(axis=1)).T # normalize
    y_m[train_index] = clamp  # clamp
```

```
#  预测准确率
predict = y_m[test_index].argmax(axis=1)
np.sum(y[test_index] == predict) / float(len(predict))
```

```
0.9166666666666663
```

Label Propagation learning a complex structure

Example of LabelPropagation learning a complex internal structure to demonstrate “manifold learning”. The outer circle should be labeled “red” and the inner circle “blue”. Because both label groups lie inside their own distinct shape, we can see that the labels propagate correctly around the circle.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.semi_supervised import label_propagation
from sklearn.datasets import make_circles

# generate ring with inner box
n_samples = 200
X, y = make_circles(n_samples=n_samples, shuffle=False)
outer, inner = 0, 1
labels = -np.ones(n_samples)
labels[0] = outer
labels[-1] = inner

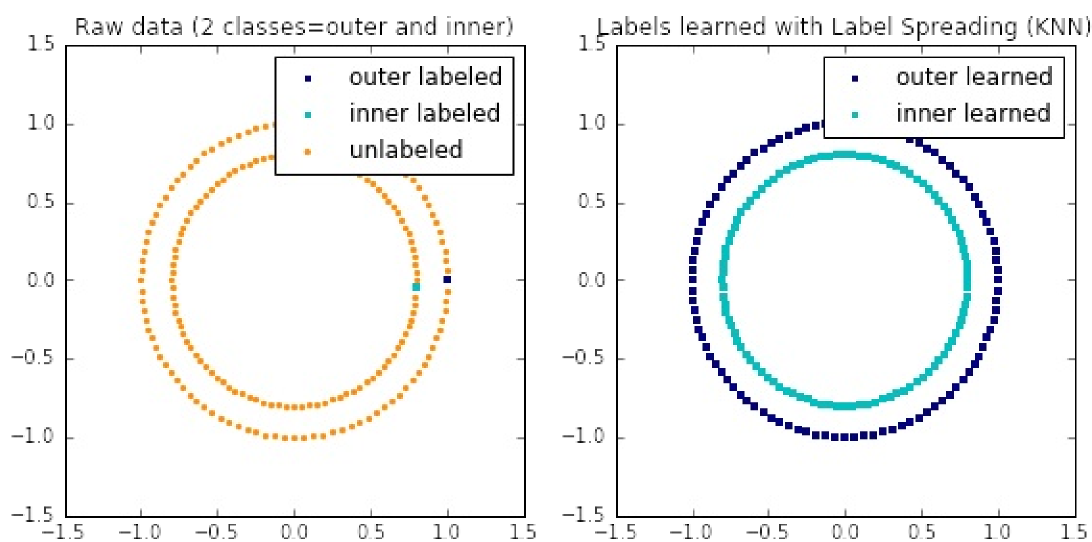
# Learn with LabelSpreading
label_spread = label_propagation.LabelSpreading(kernel='knn', al
pha=1.0)
label_spread.fit(X, labels)

# Plot output labels
output_labels = label_spread.transduction_
plt.figure(figsize=(8.5, 4))
plt.subplot(1, 2, 1)
plt.scatter(X[labels == outer, 0], X[labels == outer, 1], color=
'navy',
            marker='s', lw=0, label="outer labeled", s=10)
```

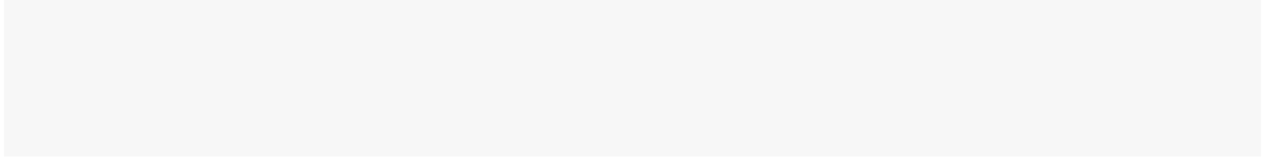
```
plt.scatter(X[labels == inner, 0], X[labels == inner, 1], color=
'c',
            marker='s', lw=0, label='inner labeled', s=10)
plt.scatter(X[labels == -1, 0], X[labels == -1, 1], color='darko
range',
            marker='.', label='unlabeled')
plt.legend(scatterpoints=1, shadow=False, loc='upper right')
plt.title("Raw data (2 classes=outer and inner)")

plt.subplot(1, 2, 2)
output_label_array = np.asarray(output_labels)
outer_numbers = np.where(output_label_array == outer)[0]
inner_numbers = np.where(output_label_array == inner)[0]
plt.scatter(X[outer_numbers, 0], X[outer_numbers, 1], color='nav
y',
            marker='s', lw=0, s=10, label="outer learned")
plt.scatter(X[inner_numbers, 0], X[inner_numbers, 1], color='c',
            marker='s', lw=0, s=10, label="inner learned")
plt.legend(scatterpoints=1, shadow=False, loc='upper right')
plt.title("Labels learned with Label Spreading (KNN)")

plt.subplots_adjust(left=0.07, bottom=0.07, right=0.93, top=0.92
)
plt.show()
```



练习：使用信贷数据集中的自变量，来对这些用户进行聚类



Sections

Obtaining the IMDb movie review dataset

[back to top]

数据可从这下载

解压之后，下面代码可将数据读成 Pandas 的 DataFrame

```
cd Documents/shanghai_python/data/aclImdb/
```

```
/Users/xiaokai/Documents/shanghai_python/data/aclImdb
```

```
ls
```

```
README      imdb.vocab  imdbEr.txt  [34mtest[m[m/      [34mtrai
n[m[m/
```

```
cd train/pos/
```

```
/Users/xiaokai/Documents/shanghai_python/data/aclImdb/train/pos
```

```
ls
```

```
0_9.txt       12250_10.txt  3250_9.txt    5500_10.txt   7751_7.t
xt
10000_8.txt   12251_9.txt   3251_7.txt    5501_10.txt   7752_8.t
xt
10001_10.txt  12252_10.txt  3252_9.txt    5502_10.txt   7753_10.
txt
10002_7.txt   12253_9.txt   3253_8.txt    5503_10.txt   7754_9.t
xt
10003_8.txt   12254_10.txt  3254_9.txt    5504_10.txt   7755_9.t
xt
10004_8.txt   12255_10.txt  3255_10.txt   5505_10.txt   7756_10.
txt
10005_7.txt   12256_9.txt   3256_9.txt    5506_10.txt   7757_9.t
xt
10006_7.txt   12257_8.txt   3257_9.txt    5507_7.txt    7758_7.t
xt
10007_7.txt   12258_10.txt  3258_10.txt   5508_10.txt   7759_10.
txt
10008_7.txt   12259_7.txt   3259_7.txt    5509_10.txt   775_7.tx
t
10009_9.txt   1225_10.txt   325_9.txt     550_10.txt    7760_10.
txt
1000_8.txt    12260_10.txt  3260_8.txt    5510_10.txt   7761_10.
txt
10010_7.txt   12261_10.txt  3261_9.txt    5511_8.txt    7762_8.t
xt
```

```
10011_9.txt    12262_7.txt    3262_8.txt    5512_7.txt    7763_8.t
xt
10012_8.txt    12263_10.txt   3263_7.txt    5513_7.txt    7764_9.t
xt
10013_7.txt    12264_9.txt    3264_8.txt    5514_9.txt    7765_7.t
xt
10014_8.txt    12265_10.txt   3265_10.txt   5515_7.txt    7766_7.t
xt
10015_8.txt    12266_9.txt    3266_7.txt    5516_7.txt    7767_8.t
xt
10016_8.txt    12267_8.txt    3267_8.txt    5517_9.txt    7768_8.t
xt
10017_9.txt    12268_7.txt    3268_8.txt    5518_8.txt    7769_7.t
xt
10018_8.txt    12269_8.txt    3269_8.txt    5519_9.txt    776_7.tx
t
10019_8.txt    1226_10.txt    326_10.txt    551_8.txt     7770_7.t
xt
1001_8.txt     12270_10.txt   3270_8.txt    5520_9.txt    7771_9.t
xt
10020_8.txt    12271_7.txt    3271_9.txt    5521_8.txt    7772_7.t
xt
10021_8.txt    12272_7.txt    3272_8.txt    5522_8.txt    7773_10.
txt
10022_7.txt    12273_10.txt   3273_9.txt    5523_8.txt    7774_7.t
xt
10023_9.txt    12274_10.txt   3274_8.txt    5524_8.txt    7775_8.t
xt
10024_9.txt    12275_9.txt    3275_10.txt   5525_8.txt    7776_7.t
xt
10025_9.txt    12276_7.txt    3276_8.txt    5526_10.txt   7777_10.
txt
10026_7.txt    12277_10.txt   3277_9.txt    5527_8.txt    7778_10.
txt
10027_7.txt    12278_10.txt   3278_8.txt    5528_7.txt    7779_7.t
xt
10028_10.txt   12279_9.txt    3279_8.txt    5529_7.txt    777_7.tx
t
10029_10.txt   1227_8.txt     327_8.txt     552_8.txt     7780_8.t
xt
```

```
1002_7.txt    12280_9.txt   3280_10.txt   5530_8.txt    7781_8.t
xt
10030_10.txt  12281_9.txt   3281_10.txt   5531_10.txt   7782_7.t
xt
10031_10.txt  12282_8.txt   3282_8.txt    5532_10.txt   7783_7.t
xt
10032_10.txt  12283_8.txt   3283_8.txt    5533_7.txt    7784_7.t
xt
10033_10.txt  12284_7.txt   3284_10.txt   5534_7.txt    7785_9.t
xt
10034_8.txt   12285_10.txt  3285_10.txt   5535_8.txt    7786_8.t
xt
10035_9.txt   12286_7.txt   3286_10.txt   5536_9.txt    7787_10.
txt
10036_8.txt   12287_10.txt  3287_9.txt    5537_10.txt   7788_10.
txt
10037_9.txt   12288_9.txt   3288_10.txt   5538_7.txt    7789_10.
txt
10038_10.txt  12289_9.txt   3289_10.txt   5539_10.txt   778_10.t
xt
10039_10.txt  1228_9.txt    328_10.txt    553_7.txt     7790_9.t
xt
1003_10.txt   12290_10.txt  3290_8.txt    5540_10.txt   7791_10.
txt
10040_10.txt  12291_10.txt  3291_8.txt    5541_9.txt    7792_10.
txt
10041_10.txt  12292_10.txt  3292_10.txt   5542_9.txt    7793_10.
txt
10042_10.txt  12293_10.txt  3293_10.txt   5543_7.txt    7794_7.t
xt
10043_10.txt  12294_8.txt   3294_9.txt    5544_7.txt    7795_9.t
xt
10044_9.txt   12295_9.txt   3295_10.txt   5545_8.txt    7796_10.
txt
10045_10.txt  12296_8.txt   3296_10.txt   5546_7.txt    7797_10.
txt
10046_9.txt   12297_7.txt   3297_10.txt   5547_8.txt    7798_10.
txt
10047_10.txt  12298_8.txt   3298_10.txt   5548_7.txt    7799_8.t
xt
```

```
10048_10.txt  12299_7.txt   3299_10.txt   5549_8.txt    779_10.t
xt
10049_8.txt   1229_8.txt    329_10.txt    554_7.txt     77_7.txt
1004_7.txt    122_9.txt     32_10.txt     5550_7.txt    7800_8.t
xt
10050_10.txt  12300_10.txt  3300_9.txt    5551_7.txt    7801_9.t
xt
10051_10.txt  12301_8.txt   3301_10.txt   5552_8.txt    7802_10.
txt
10052_10.txt  12302_7.txt   3302_10.txt   5553_8.txt    7803_10.
txt
10053_8.txt   12303_9.txt   3303_10.txt   5554_8.txt    7804_10.
txt
10054_10.txt  12304_10.txt  3304_10.txt   5555_10.txt   7805_10.
txt
10055_7.txt   12305_8.txt   3305_10.txt   5556_10.txt   7806_10.
txt
10056_8.txt   12306_9.txt   3306_8.txt    5557_9.txt    7807_10.
txt
10057_9.txt   12307_10.txt  3307_8.txt    5558_7.txt    7808_10.
txt
10058_7.txt   12308_10.txt  3308_8.txt    5559_7.txt    7809_10.
txt
10059_10.txt  12309_10.txt  3309_9.txt    555_8.txt     780_7.tx
t
1005_10.txt   1230_10.txt   330_10.txt    5560_7.txt    7810_7.t
xt
10060_9.txt   12310_10.txt  3310_8.txt    5561_8.txt    7811_7.t
xt
10061_8.txt   12311_9.txt   3311_10.txt   5562_10.txt   7812_8.t
xt
10062_10.txt  12312_10.txt  3312_8.txt    5563_8.txt    7813_10.
txt
10063_9.txt   12313_10.txt  3313_10.txt   5564_10.txt   7814_10.
txt
10064_10.txt  12314_10.txt  3314_10.txt   5565_8.txt    7815_8.t
xt
10065_9.txt   12315_9.txt   3315_10.txt   5566_9.txt    7816_9.t
xt
10066_10.txt  12316_10.txt  3316_10.txt   5567_7.txt    7817_10.
```

```
txt
10067_9.txt   12317_10.txt  3317_10.txt   5568_8.txt    7818_9.t
xt
10068_8.txt   12318_9.txt   3318_9.txt    5569_7.txt    7819_9.t
xt
10069_8.txt   12319_8.txt   3319_9.txt    556_9.txt     781_9.tx
t
1006_8.txt    1231_7.txt    331_10.txt    5570_10.txt   7820_8.t
xt
10070_9.txt   12320_8.txt   3320_8.txt    5571_10.txt   7821_8.t
xt
10071_9.txt   12321_10.txt  3321_7.txt    5572_10.txt   7822_8.t
xt
10072_9.txt   12322_10.txt  3322_8.txt    5573_10.txt   7823_7.t
xt
10073_10.txt  12323_10.txt  3323_9.txt    5574_10.txt   7824_10.
txt
10074_9.txt   12324_10.txt  3324_7.txt    5575_8.txt    7825_9.t
xt
10075_9.txt   12325_9.txt   3325_7.txt    5576_9.txt    7826_8.t
xt
10076_9.txt   12326_10.txt  3326_8.txt    5577_8.txt    7827_10.
txt
10077_10.txt  12327_9.txt   3327_10.txt   5578_10.txt   7828_10.
txt
10078_8.txt   12328_10.txt  3328_7.txt    5579_8.txt    7829_7.t
xt
10079_8.txt   12329_10.txt  3329_7.txt    557_9.txt     782_9.tx
t
1007_10.txt   1232_10.txt   332_10.txt    5580_7.txt    7830_10.
txt
10080_10.txt  12330_7.txt   3330_7.txt    5581_7.txt    7831_8.t
xt
10081_9.txt   12331_8.txt   3331_8.txt    5582_9.txt    7832_7.t
xt
10082_10.txt  12332_8.txt   3332_8.txt    5583_8.txt    7833_8.t
xt
10083_7.txt   12333_9.txt   3333_8.txt    5584_8.txt    7834_8.t
xt
10084_10.txt  12334_7.txt   3334_9.txt    5585_7.txt    7835_9.t
```

```
xt
10085_10.txt  12335_9.txt   3335_8.txt    5586_8.txt    7836_8.t
xt
10086_7.txt   12336_9.txt   3336_7.txt    5587_7.txt    7837_10.
txt
10087_10.txt  12337_7.txt   3337_9.txt    5588_10.txt   7838_8.t
xt
10088_10.txt  12338_10.txt  3338_7.txt    5589_7.txt    7839_10.
txt
10089_7.txt   12339_10.txt  3339_7.txt    558_10.txt    783_10.t
xt
1008_10.txt   1233_7.txt    333_10.txt    5590_8.txt    7840_8.t
xt
10090_8.txt   12340_8.txt   3340_8.txt    5591_8.txt    7841_10.
txt
10091_7.txt   12341_7.txt   3341_7.txt    5592_10.txt   7842_10.
txt
10092_8.txt   12342_8.txt   3342_7.txt    5593_10.txt   7843_9.t
xt
10093_7.txt   12343_7.txt   3343_7.txt    5594_10.txt   7844_10.
txt
10094_7.txt   12344_8.txt   3344_9.txt    5595_7.txt    7845_10.
txt
10095_7.txt   12345_10.txt  3345_9.txt    5596_10.txt   7846_10.
txt
10096_7.txt   12346_10.txt  3346_7.txt    5597_10.txt   7847_9.t
xt
10097_9.txt   12347_9.txt   3347_7.txt    5598_8.txt    7848_10.
txt
10098_10.txt  12348_9.txt   3348_7.txt    5599_7.txt    7849_9.t
xt
10099_10.txt  12349_10.txt  3349_8.txt    559_8.txt     784_10.t
xt
1009_8.txt    1234_10.txt   334_10.txt    55_9.txt      7850_8.t
xt
100_7.txt     12350_9.txt   3350_7.txt    5600_8.txt    7851_8.t
xt
10100_10.txt  12351_8.txt   3351_8.txt    5601_8.txt    7852_9.t
xt
10101_8.txt   12352_8.txt   3352_9.txt    5602_10.txt   7853_9.t
```

```
xt
10102_7.txt   12353_9.txt   3353_8.txt    5603_7.txt    7854_10.
txt
10103_8.txt   12354_10.txt  3354_10.txt   5604_8.txt    7855_8.t
xt
10104_10.txt  12355_10.txt  3355_10.txt   5605_7.txt    7856_10.
txt
10105_8.txt   12356_8.txt   3356_10.txt   5606_8.txt    7857_10.
txt
10106_8.txt   12357_7.txt   3357_9.txt    5607_7.txt    7858_10.
txt
10107_8.txt   12358_7.txt   3358_9.txt    5608_9.txt    7859_7.t
xt
10108_10.txt  12359_8.txt   3359_10.txt   5609_10.txt   785_9.tx
t
10109_10.txt  1235_10.txt   335_10.txt    560_8.txt     7860_10.
txt
1010_10.txt   12360_10.txt  3360_7.txt    5610_7.txt    7861_10.
txt
10110_10.txt  12361_7.txt   3361_7.txt    5611_10.txt   7862_7.t
xt
10111_7.txt   12362_8.txt   3362_8.txt    5612_8.txt    7863_10.
txt
10112_7.txt   12363_9.txt   3363_10.txt   5613_9.txt    7864_8.t
xt
10113_10.txt  12364_7.txt   3364_10.txt   5614_10.txt   7865_8.t
xt
10114_10.txt  12365_7.txt   3365_9.txt    5615_8.txt    7866_10.
txt
10115_10.txt  12366_10.txt  3366_10.txt   5616_8.txt    7867_7.t
xt
10116_10.txt  12367_9.txt   3367_9.txt    5617_8.txt    7868_8.t
xt
10117_8.txt   12368_8.txt   3368_8.txt    5618_10.txt   7869_9.t
xt
10118_7.txt   12369_7.txt   3369_7.txt    5619_9.txt    786_7.tx
t
10119_7.txt   1236_7.txt    336_10.txt    561_10.txt    7870_10.
txt
1011_10.txt   12370_8.txt   3370_9.txt    5620_10.txt   7871_9.t
```

```
xt
10120_7.txt   12371_8.txt   3371_8.txt    5621_10.txt   7872_10.
txt
10121_8.txt   12372_7.txt   3372_9.txt    5622_10.txt   7873_8.t
xt
10122_7.txt   12373_7.txt   3373_7.txt    5623_10.txt   7874_8.t
xt
10123_10.txt  12374_7.txt   3374_7.txt    5624_7.txt    7875_7.t
xt
10124_8.txt   12375_7.txt   3375_9.txt    5625_8.txt    7876_8.t
xt
10125_8.txt   12376_7.txt   3376_9.txt    5626_8.txt    7877_7.t
xt
10126_10.txt  12377_10.txt  3377_8.txt    5627_10.txt   7878_7.t
xt
10127_8.txt   12378_8.txt   3378_7.txt    5628_7.txt    7879_7.t
xt
10128_9.txt   12379_8.txt   3379_7.txt    5629_10.txt   787_9.tx
t
10129_7.txt   1237_8.txt    337_9.txt     562_8.txt     7880_8.t
xt
1012_10.txt   12380_7.txt   3380_7.txt    5630_8.txt    7881_9.t
xt
10130_10.txt  12381_8.txt   3381_10.txt   5631_8.txt    7882_7.t
xt
10131_10.txt  12382_8.txt   3382_10.txt   5632_8.txt    7883_10.
txt
10132_9.txt   12383_7.txt   3383_8.txt    5633_8.txt    7884_7.t
xt
10133_7.txt   12384_8.txt   3384_8.txt    5634_8.txt    7885_9.t
xt
10134_7.txt   12385_7.txt   3385_9.txt    5635_7.txt    7886_10.
txt
10135_7.txt   12386_8.txt   3386_9.txt    5636_10.txt   7887_7.t
xt
10136_7.txt   12387_10.txt  3387_10.txt   5637_10.txt   7888_8.t
xt
10137_7.txt   12388_8.txt   3388_7.txt    5638_10.txt   7889_10.
txt
10138_8.txt   12389_10.txt  3389_7.txt    5639_7.txt    788_8.tx
```

```
t
10139_8.txt    1238_7.txt    338_10.txt    563_10.txt    7890_7.t
xt
1013_9.txt     12390_8.txt   3390_8.txt    5640_7.txt    7891_8.t
xt
10140_8.txt    12391_9.txt   3391_7.txt    5641_7.txt    7892_7.t
xt
10141_9.txt    12392_9.txt   3392_7.txt    5642_7.txt    7893_7.t
xt
10142_8.txt    12393_8.txt   3393_9.txt    5643_7.txt    7894_10.
txt
10143_8.txt    12394_10.txt  3394_10.txt   5644_8.txt    7895_10.
txt
10144_8.txt    12395_8.txt   3395_9.txt    5645_10.txt   7896_10.
txt
10145_8.txt    12396_7.txt   3396_7.txt    5646_10.txt   7897_8.t
xt
10146_7.txt    12397_8.txt   3397_10.txt   5647_10.txt   7898_10.
txt
10147_10.txt   12398_8.txt   3398_10.txt   5648_10.txt   7899_10.
txt
10148_10.txt   12399_9.txt   3399_10.txt   5649_10.txt   789_7.tx
t
10149_9.txt    1239_8.txt    339_10.txt    564_8.txt     78_10.tx
t
1014_9.txt     123_10.txt    33_7.txt      5650_10.txt   7900_10.
txt
10150_9.txt    12400_8.txt   3400_8.txt    5651_10.txt   7901_7.t
xt
10151_8.txt    12401_8.txt   3401_8.txt    5652_10.txt   7902_10.
txt
10152_9.txt    12402_7.txt   3402_8.txt    5653_10.txt   7903_10.
txt
10153_9.txt    12403_8.txt   3403_9.txt    5654_8.txt    7904_8.t
xt
10154_8.txt    12404_9.txt   3404_8.txt    5655_7.txt    7905_10.
txt
10155_9.txt    12405_10.txt  3405_9.txt    5656_10.txt   7906_7.t
xt
10156_10.txt   12406_8.txt   3406_10.txt   5657_9.txt    7907_7.t
```

```
xt
10157_10.txt  12407_7.txt   3407_8.txt    5658_10.txt   7908_9.t
xt
10158_7.txt   12408_7.txt   3408_10.txt   5659_9.txt    7909_10.
txt
10159_7.txt   12409_7.txt   3409_10.txt   565_10.txt    790_9.tx
t
1015_10.txt   1240_7.txt    340_10.txt    5660_10.txt   7910_10.
txt
10160_7.txt   12410_8.txt   3410_9.txt    5661_10.txt   7911_10.
txt
10161_9.txt   12411_7.txt   3411_9.txt    5662_10.txt   7912_8.t
xt
10162_9.txt   12412_9.txt   3412_8.txt    5663_10.txt   7913_10.
txt
10163_8.txt   12413_9.txt   3413_10.txt   5664_8.txt    7914_10.
txt
10164_7.txt   12414_10.txt  3414_8.txt    5665_9.txt    7915_8.t
xt
10165_7.txt   12415_8.txt   3415_9.txt    5666_10.txt   7916_8.t
xt
10166_7.txt   12416_10.txt  3416_10.txt   5667_10.txt   7917_8.t
xt
10167_7.txt   12417_10.txt  3417_10.txt   5668_10.txt   7918_10.
txt
10168_8.txt   12418_10.txt  3418_10.txt   5669_10.txt   7919_10.
txt
10169_7.txt   12419_10.txt  3419_10.txt   566_8.txt     791_9.tx
t
1016_8.txt    1241_7.txt    341_10.txt    5670_8.txt    7920_7.t
xt
10170_8.txt   12420_9.txt   3420_10.txt   5671_9.txt    7921_10.
txt
10171_7.txt   12421_10.txt  3421_10.txt   5672_9.txt    7922_7.t
xt
10172_8.txt   12422_8.txt   3422_10.txt   5673_9.txt    7923_8.t
xt
10173_8.txt   12423_10.txt  3423_7.txt    5674_7.txt    7924_9.t
xt
10174_7.txt   12424_9.txt   3424_10.txt   5675_9.txt    7925_7.t
```

```
xt
10175_10.txt  12425_7.txt   3425_8.txt    5676_9.txt    7926_10.
txt
10176_7.txt   12426_7.txt   3426_7.txt    5677_10.txt   7927_10.
txt
10177_9.txt   12427_7.txt   3427_8.txt    5678_10.txt   7928_10.
txt
10178_10.txt  12428_8.txt   3428_9.txt    5679_10.txt   7929_10.
txt
10179_9.txt   12429_7.txt   3429_10.txt   567_10.txt    792_8.tx
t
1017_8.txt    1242_9.txt    342_10.txt    5680_10.txt   7930_7.t
xt
10180_7.txt   12430_7.txt   3430_10.txt   5681_8.txt    7931_8.t
xt
10181_8.txt   12431_8.txt   3431_10.txt   5682_7.txt    7932_8.t
xt
10182_8.txt   12432_8.txt   3432_8.txt    5683_10.txt   7933_10.
txt
10183_7.txt   12433_8.txt   3433_9.txt    5684_10.txt   7934_7.t
xt
10184_9.txt   12434_10.txt  3434_8.txt    5685_10.txt   7935_8.t
xt
10185_10.txt  12435_8.txt   3435_10.txt   5686_9.txt    7936_7.t
xt
10186_8.txt   12436_7.txt   3436_8.txt    5687_7.txt    7937_10.
txt
10187_7.txt   12437_7.txt   3437_8.txt    5688_8.txt    7938_7.t
xt
10188_8.txt   12438_8.txt   3438_10.txt   5689_10.txt   7939_8.t
xt
10189_7.txt   12439_8.txt   3439_9.txt    568_7.txt     793_9.tx
t
1018_8.txt    1243_9.txt    343_10.txt    5690_7.txt    7940_7.t
xt
10190_7.txt   12440_7.txt   3440_8.txt    5691_8.txt    7941_8.t
xt
10191_10.txt  12441_9.txt   3441_8.txt    5692_7.txt    7942_10.
txt
10192_8.txt   12442_8.txt   3442_10.txt   5693_7.txt    7943_10.
```

```
txt
10193_9.txt   12443_9.txt   3443_9.txt   5694_9.txt   7944_9.t
xt
10194_10.txt  12444_10.txt  3444_10.txt  5695_8.txt   7945_8.t
xt
10195_8.txt   12445_9.txt   3445_7.txt   5696_8.txt   7946_10.
txt
10196_10.txt  12446_8.txt   3446_7.txt   5697_8.txt   7947_10.
txt
10197_7.txt   12447_8.txt   3447_9.txt   5698_8.txt   7948_7.t
xt
10198_8.txt   12448_10.txt  3448_7.txt   5699_8.txt   7949_10.
txt
10199_7.txt   12449_8.txt   3449_8.txt   569_10.txt   794_8.tx
t
1019_10.txt   1244_8.txt    344_8.txt    56_10.txt    7950_9.t
xt
101_8.txt     12450_7.txt   3450_7.txt   5700_8.txt   7951_10.
txt
10200_10.txt  12451_10.txt  3451_8.txt   5701_8.txt   7952_8.t
xt
10201_10.txt  12452_8.txt   3452_8.txt   5702_10.txt  7953_9.t
xt
10202_10.txt  12453_7.txt   3453_7.txt   5703_7.txt   7954_7.t
xt
10203_10.txt  12454_7.txt   3454_8.txt   5704_7.txt   7955_10.
txt
10204_8.txt   12455_9.txt   3455_10.txt  5705_10.txt  7956_10.
txt
10205_10.txt  12456_10.txt  3456_9.txt   5706_9.txt   7957_10.
txt
10206_10.txt  12457_8.txt   3457_7.txt   5707_10.txt  7958_10.
txt
10207_10.txt  12458_7.txt   3458_10.txt  5708_10.txt  7959_10.
txt
10208_7.txt   12459_10.txt  3459_8.txt   5709_10.txt  795_8.tx
t
10209_7.txt   1245_7.txt    345_7.txt    570_10.txt   7960_10.
txt
1020_10.txt   12460_7.txt   3460_8.txt   5710_8.txt   7961_10.
```

```
txt
10210_7.txt   12461_9.txt   3461_9.txt    5711_8.txt    7962_9.t
xt
10211_7.txt   12462_7.txt   3462_7.txt    5712_8.txt    7963_9.t
xt
10212_8.txt   12463_8.txt   3463_7.txt    5713_10.txt   7964_10.
txt
10213_8.txt   12464_10.txt  3464_10.txt   5714_7.txt    7965_10.
txt
10214_10.txt  12465_9.txt   3465_9.txt    5715_10.txt   7966_9.t
xt
10215_10.txt  12466_7.txt   3466_7.txt    5716_8.txt    7967_9.t
xt
10216_8.txt   12467_7.txt   3467_7.txt    5717_9.txt    7968_10.
txt
10217_9.txt   12468_7.txt   3468_10.txt   5718_7.txt    7969_10.
txt
10218_8.txt   12469_10.txt  3469_10.txt   5719_9.txt    796_8.tx
t
10219_10.txt  1246_8.txt    346_10.txt    571_10.txt    7970_10.
txt
1021_10.txt   12470_10.txt  3470_8.txt    5720_10.txt   7971_10.
txt
10220_7.txt   12471_7.txt   3471_8.txt    5721_10.txt   7972_10.
txt
10221_8.txt   12472_10.txt  3472_10.txt   5722_8.txt    7973_10.
txt
10222_9.txt   12473_10.txt  3473_9.txt    5723_7.txt    7974_7.t
xt
10223_10.txt  12474_9.txt   3474_10.txt   5724_10.txt   7975_8.t
xt
10224_10.txt  12475_10.txt  3475_9.txt    5725_9.txt    7976_7.t
xt
10225_9.txt   12476_10.txt  3476_10.txt   5726_7.txt    7977_8.t
xt
10226_10.txt  12477_10.txt  3477_10.txt   5727_9.txt    7978_8.t
xt
10227_10.txt  12478_9.txt   3478_8.txt    5728_10.txt   7979_9.t
xt
10228_8.txt   12479_10.txt  3479_9.txt    5729_9.txt    797_8.tx
```

```
t
10229_8.txt   1247_8.txt    347_10.txt    572_9.txt     7980_10.
txt
1022_10.txt   12480_10.txt  3480_10.txt   5730_10.txt   7981_10.
txt
10230_9.txt   12481_8.txt   3481_10.txt   5731_7.txt    7982_10.
txt
10231_10.txt  12482_8.txt   3482_10.txt   5732_7.txt    7983_9.t
xt
10232_10.txt  12483_9.txt   3483_9.txt    5733_7.txt    7984_10.
txt
10233_7.txt   12484_8.txt   3484_10.txt   5734_9.txt    7985_10.
txt
10234_10.txt  12485_8.txt   3485_9.txt    5735_7.txt    7986_10.
txt
10235_8.txt   12486_7.txt   3486_10.txt   5736_7.txt    7987_8.t
xt
10236_8.txt   12487_10.txt  3487_9.txt    5737_7.txt    7988_10.
txt
10237_10.txt  12488_8.txt   3488_7.txt    5738_10.txt   7989_10.
txt
10238_10.txt  12489_10.txt  3489_10.txt   5739_10.txt   798_10.t
xt
10239_10.txt  1248_8.txt    348_7.txt     573_9.txt     7990_7.t
xt
1023_10.txt   12490_8.txt   3490_8.txt    5740_10.txt   7991_7.t
xt
10240_8.txt   12491_8.txt   3491_8.txt    5741_10.txt   7992_7.t
xt
10241_8.txt   12492_7.txt   3492_7.txt    5742_10.txt   7993_10.
txt
10242_8.txt   12493_8.txt   3493_8.txt    5743_10.txt   7994_9.t
xt
10243_10.txt  12494_8.txt   3494_9.txt    5744_7.txt    7995_8.t
xt
10244_7.txt   12495_7.txt   3495_7.txt    5745_8.txt    7996_8.t
xt
10245_10.txt  12496_8.txt   3496_8.txt    5746_9.txt    7997_10.
txt
10246_10.txt  12497_10.txt  3497_10.txt   5747_8.txt    7998_9.t
```

```
xt
10247_10.txt  12498_10.txt  3498_10.txt   5748_10.txt   7999_10.
txt
10248_7.txt   12499_7.txt   3499_8.txt    5749_10.txt   799_8.tx
t
10249_7.txt   1249_9.txt    349_10.txt    574_7.txt     79_10.tx
t
1024_9.txt    124_10.txt    34_8.txt      5750_8.txt    7_7.txt
10250_10.txt  1250_10.txt   3500_10.txt   5751_10.txt   8000_10.
txt
10251_10.txt  1251_9.txt    3501_9.txt    5752_8.txt    8001_10.
txt
10252_9.txt   1252_8.txt    3502_9.txt    5753_8.txt    8002_7.t
xt
10253_10.txt  1253_10.txt   3503_7.txt    5754_9.txt    8003_8.t
xt
10254_8.txt   1254_7.txt    3504_7.txt    5755_7.txt    8004_9.t
xt
10255_9.txt   1255_10.txt   3505_8.txt    5756_8.txt    8005_9.t
xt
10256_8.txt   1256_8.txt    3506_10.txt   5757_8.txt    8006_10.
txt
10257_8.txt   1257_10.txt   3507_7.txt    5758_10.txt   8007_7.t
xt
10258_10.txt  1258_9.txt    3508_8.txt    5759_10.txt   8008_7.t
xt
10259_8.txt   1259_9.txt    3509_10.txt   575_10.txt    8009_9.t
xt
1025_8.txt    125_7.txt     350_9.txt     5760_8.txt    800_9.tx
t
10260_10.txt  1260_10.txt   3510_7.txt    5761_8.txt    8010_7.t
xt
10261_8.txt   1261_8.txt    3511_8.txt    5762_8.txt    8011_10.
txt
10262_10.txt  1262_8.txt    3512_7.txt    5763_8.txt    8012_8.t
xt
10263_10.txt  1263_8.txt    3513_10.txt   5764_7.txt    8013_8.t
xt
10264_10.txt  1264_10.txt   3514_7.txt    5765_9.txt    8014_7.t
xt
```

```
10265_9.txt    1265_10.txt    3515_7.txt    5766_10.txt    8015_10.
txt
10266_9.txt    1266_8.txt     3516_8.txt    5767_8.txt     8016_10.
txt
10267_8.txt    1267_7.txt     3517_8.txt    5768_8.txt     8017_9.t
xt
10268_9.txt    1268_7.txt     3518_10.txt   5769_9.txt     8018_10.
txt
10269_7.txt    1269_8.txt     3519_7.txt    576_10.txt     8019_7.t
xt
1026_9.txt     126_10.txt     351_10.txt    5770_10.txt    801_8.tx
t
10270_9.txt    1270_7.txt     3520_9.txt    5771_8.txt     8020_8.t
xt
10271_10.txt   1271_7.txt     3521_9.txt    5772_10.txt    8021_7.t
xt
10272_10.txt   1272_7.txt     3522_8.txt    5773_9.txt     8022_8.t
xt
10273_8.txt    1273_7.txt     3523_9.txt    5774_8.txt     8023_7.t
xt
10274_8.txt    1274_7.txt     3524_9.txt    5775_7.txt     8024_10.
txt
10275_10.txt   1275_8.txt     3525_7.txt    5776_10.txt    8025_9.t
xt
10276_10.txt   1276_9.txt     3526_8.txt    5777_8.txt     8026_9.t
xt
10277_9.txt    1277_10.txt    3527_7.txt    5778_9.txt     8027_10.
txt
10278_7.txt    1278_9.txt     3528_7.txt    5779_10.txt    8028_10.
txt
10279_8.txt    1279_9.txt     3529_9.txt    577_8.txt      8029_10.
txt
1027_8.txt     127_7.txt      352_10.txt    5780_10.txt    802_10.t
xt
10280_10.txt   1280_10.txt    3530_7.txt    5781_10.txt    8030_9.t
xt
10281_7.txt    1281_10.txt    3531_7.txt    5782_10.txt    8031_9.t
xt
10282_8.txt    1282_9.txt     3532_8.txt    5783_7.txt     8032_10.
txt
```

```
10283_10.txt  1283_10.txt  3533_10.txt  5784_8.txt   8033_7.t
xt
10284_9.txt   1284_7.txt   3534_10.txt  5785_10.txt  8034_10.
txt
10285_10.txt  1285_9.txt   3535_8.txt   5786_10.txt  8035_7.t
xt
10286_9.txt   1286_8.txt   3536_10.txt  5787_9.txt   8036_8.t
xt
10287_8.txt   1287_9.txt   3537_10.txt  5788_8.txt   8037_10.
txt
10288_10.txt  1288_8.txt   3538_10.txt  5789_9.txt   8038_7.t
xt
10289_10.txt  1289_7.txt   3539_9.txt   578_10.txt   8039_8.t
xt
1028_10.txt   128_7.txt    353_9.txt    5790_7.txt   803_10.t
xt
10290_8.txt   1290_7.txt   3540_8.txt   5791_9.txt   8040_9.t
xt
10291_7.txt   1291_10.txt  3541_7.txt   5792_10.txt  8041_7.t
xt
10292_7.txt   1292_8.txt   3542_8.txt   5793_10.txt  8042_7.t
xt
10293_8.txt   1293_8.txt   3543_10.txt  5794_10.txt  8043_9.t
xt
10294_8.txt   1294_7.txt   3544_9.txt   5795_10.txt  8044_10.
txt
10295_7.txt   1295_10.txt  3545_7.txt   5796_8.txt   8045_8.t
xt
10296_8.txt   1296_10.txt  3546_10.txt  5797_8.txt   8046_9.t
xt
10297_8.txt   1297_8.txt   3547_8.txt   5798_8.txt   8047_9.t
xt
10298_9.txt   1298_7.txt   3548_8.txt   5799_10.txt  8048_7.t
xt
10299_9.txt   1299_10.txt  3549_8.txt   579_10.txt   8049_7.t
xt
1029_9.txt    129_9.txt    354_9.txt    57_10.txt    804_10.t
xt
102_10.txt    12_9.txt     3550_10.txt  5800_10.txt  8050_10.
txt
```

```
10300_10.txt  1300_10.txt  3551_8.txt   5801_10.txt  8051_7.t
xt
10301_8.txt   1301_10.txt  3552_10.txt  5802_10.txt  8052_9.t
xt
10302_9.txt   1302_10.txt  3553_8.txt   5803_10.txt  8053_8.t
xt
10303_7.txt   1303_10.txt  3554_10.txt  5804_10.txt  8054_8.t
xt
10304_7.txt   1304_7.txt   3555_7.txt   5805_9.txt   8055_8.t
xt
10305_8.txt   1305_10.txt  3556_10.txt  5806_7.txt   8056_8.t
xt
10306_8.txt   1306_7.txt   3557_9.txt   5807_8.txt   8057_10.
txt
10307_8.txt   1307_10.txt  3558_8.txt   5808_10.txt  8058_8.t
xt
10308_8.txt   1308_9.txt   3559_10.txt  5809_10.txt  8059_10.
txt
10309_7.txt   1309_10.txt  355_9.txt    580_9.txt    805_9.tx
t
1030_10.txt   130_9.txt    3560_8.txt   5810_9.txt   8060_9.t
xt
10310_9.txt   1310_7.txt   3561_9.txt   5811_9.txt   8061_10.
txt
10311_9.txt   1311_10.txt  3562_8.txt   5812_9.txt   8062_8.t
xt
10312_10.txt  1312_9.txt   3563_8.txt   5813_8.txt   8063_10.
txt
10313_7.txt   1313_9.txt   3564_10.txt  5814_8.txt   8064_10.
txt
10314_8.txt   1314_7.txt   3565_10.txt  5815_10.txt  8065_9.t
xt
10315_8.txt   1315_10.txt  3566_9.txt   5816_10.txt  8066_9.t
xt
10316_8.txt   1316_9.txt   3567_8.txt   5817_10.txt  8067_8.t
xt
10317_7.txt   1317_8.txt   3568_7.txt   5818_7.txt   8068_9.t
xt
10318_7.txt   1318_8.txt   3569_8.txt   5819_10.txt  8069_8.t
xt
```

```
10319_7.txt   1319_10.txt   356_10.txt    581_10.txt    806_10.t
xt
1031_10.txt   131_10.txt    3570_10.txt   5820_7.txt    8070_8.t
xt
10320_7.txt   1320_10.txt   3571_7.txt    5821_7.txt    8071_9.t
xt
10321_10.txt  1321_10.txt   3572_10.txt   5822_10.txt   8072_10.
txt
10322_7.txt   1322_8.txt    3573_7.txt    5823_10.txt   8073_8.t
xt
10323_10.txt  1323_10.txt   3574_10.txt   5824_10.txt   8074_10.
txt
10324_9.txt   1324_7.txt    3575_10.txt   5825_9.txt    8075_8.t
xt
10325_10.txt  1325_9.txt    3576_9.txt    5826_10.txt   8076_7.t
xt
10326_10.txt  1326_10.txt   3577_7.txt    5827_10.txt   8077_7.t
xt
10327_7.txt   1327_9.txt    3578_7.txt    5828_9.txt    8078_10.
txt
10328_8.txt   1328_10.txt   3579_8.txt    5829_10.txt   8079_8.t
xt
10329_8.txt   1329_8.txt    357_10.txt    582_9.txt     807_10.t
xt
1032_7.txt    132_9.txt     3580_10.txt   5830_8.txt    8080_8.t
xt
10330_8.txt   1330_9.txt    3581_9.txt    5831_8.txt    8081_9.t
xt
10331_10.txt  1331_7.txt    3582_8.txt    5832_9.txt    8082_8.t
xt
10332_8.txt   1332_8.txt    3583_10.txt   5833_10.txt   8083_7.t
xt
10333_8.txt   1333_10.txt   3584_10.txt   5834_9.txt    8084_7.t
xt
10334_8.txt   1334_8.txt    3585_9.txt    5835_10.txt   8085_7.t
xt
10335_8.txt   1335_10.txt   3586_10.txt   5836_8.txt    8086_9.t
xt
10336_8.txt   1336_7.txt    3587_10.txt   5837_7.txt    8087_8.t
xt
```

```
10337_9.txt    1337_7.txt     3588_8.txt     5838_8.txt     8088_7.t
xt
10338_9.txt    1338_7.txt     3589_8.txt     5839_7.txt     8089_10.
txt
10339_7.txt    1339_9.txt     358_10.txt     583_8.txt      808_9.tx
t
1033_10.txt    133_10.txt     3590_8.txt     5840_7.txt     8090_9.t
xt
10340_9.txt    1340_10.txt    3591_9.txt     5841_7.txt     8091_9.t
xt
10341_7.txt    1341_10.txt    3592_10.txt    5842_8.txt     8092_8.t
xt
10342_7.txt    1342_10.txt    3593_8.txt     5843_10.txt    8093_7.t
xt
10343_7.txt    1343_9.txt     3594_7.txt     5844_8.txt     8094_9.t
xt
10344_7.txt    1344_9.txt     3595_10.txt    5845_7.txt     8095_8.t
xt
10345_7.txt    1345_9.txt     3596_8.txt     5846_10.txt    8096_10.
txt
10346_9.txt    1346_9.txt     3597_10.txt    5847_10.txt    8097_7.t
xt
10347_9.txt    1347_10.txt    3598_10.txt    5848_7.txt     8098_7.t
xt
10348_8.txt    1348_10.txt    3599_8.txt     5849_8.txt     8099_7.t
xt
10349_10.txt   1349_8.txt     359_8.txt      584_8.txt      809_10.t
xt
1034_7.txt     134_10.txt     35_8.txt       5850_9.txt     80_9.txt
10350_10.txt   1350_9.txt     3600_7.txt     5851_10.txt    8100_10.
txt
10351_8.txt    1351_10.txt    3601_7.txt     5852_7.txt     8101_8.t
xt
10352_10.txt   1352_10.txt    3602_7.txt     5853_10.txt    8102_7.t
xt
10353_9.txt    1353_10.txt    3603_7.txt     5854_8.txt     8103_10.
txt
10354_9.txt    1354_9.txt     3604_10.txt    5855_9.txt     8104_7.t
xt
10355_9.txt    1355_8.txt     3605_8.txt     5856_8.txt     8105_8.t
```

```
xt
10356_9.txt   1356_8.txt    3606_9.txt    5857_10.txt   8106_7.t
xt
10357_8.txt   1357_10.txt   3607_8.txt    5858_8.txt    8107_8.t
xt
10358_9.txt   1358_10.txt   3608_9.txt    5859_9.txt    8108_10.
txt
10359_7.txt   1359_7.txt    3609_8.txt    585_10.txt    8109_10.
txt
1035_7.txt    135_7.txt     360_10.txt    5860_8.txt    810_10.t
xt
10360_8.txt   1360_10.txt   3610_10.txt   5861_8.txt    8110_9.t
xt
10361_7.txt   1361_10.txt   3611_10.txt   5862_8.txt    8111_8.t
xt
10362_8.txt   1362_10.txt   3612_7.txt    5863_8.txt    8112_9.t
xt
10363_9.txt   1363_10.txt   3613_8.txt    5864_7.txt    8113_8.t
xt
10364_10.txt  1364_8.txt    3614_7.txt    5865_9.txt    8114_8.t
xt
10365_8.txt   1365_8.txt    3615_9.txt    5866_9.txt    8115_10.
txt
10366_10.txt  1366_8.txt    3616_7.txt    5867_9.txt    8116_9.t
xt
10367_8.txt   1367_10.txt   3617_8.txt    5868_8.txt    8117_8.t
xt
10368_7.txt   1368_10.txt   3618_7.txt    5869_10.txt   8118_10.
txt
10369_8.txt   1369_8.txt    3619_8.txt    586_10.txt    8119_10.
txt
1036_9.txt    136_10.txt    361_10.txt    5870_8.txt    811_10.t
xt
10370_9.txt   1370_8.txt    3620_10.txt   5871_9.txt    8120_7.t
xt
10371_8.txt   1371_8.txt    3621_8.txt    5872_9.txt    8121_8.t
xt
10372_7.txt   1372_7.txt    3622_8.txt    5873_8.txt    8122_10.
txt
10373_7.txt   1373_10.txt   3623_8.txt    5874_10.txt   8123_8.t
```

```
xt
10374_8.txt   1374_8.txt    3624_10.txt   5875_8.txt    8124_9.t
xt
10375_10.txt  1375_8.txt    3625_8.txt    5876_8.txt    8125_7.t
xt
10376_7.txt   1376_7.txt    3626_7.txt    5877_8.txt    8126_7.t
xt
10377_9.txt   1377_7.txt    3627_8.txt    5878_10.txt   8127_8.t
xt
10378_8.txt   1378_9.txt    3628_8.txt    5879_10.txt   8128_10.
txt
10379_10.txt  1379_8.txt    3629_8.txt    587_10.txt    8129_9.t
xt
1037_8.txt    137_7.txt     362_10.txt    5880_10.txt   812_10.t
xt
10380_10.txt  1380_7.txt    3630_7.txt    5881_7.txt    8130_10.
txt
10381_10.txt  1381_7.txt    3631_7.txt    5882_9.txt    8131_8.t
xt
10382_10.txt  1382_8.txt    3632_7.txt    5883_10.txt   8132_10.
txt
10383_10.txt  1383_7.txt    3633_10.txt   5884_10.txt   8133_7.t
xt
10384_10.txt  1384_7.txt    3634_8.txt    5885_10.txt   8134_10.
txt
10385_10.txt  1385_8.txt    3635_9.txt    5886_10.txt   8135_10.
txt
10386_8.txt   1386_8.txt    3636_8.txt    5887_7.txt    8136_9.t
xt
10387_7.txt   1387_8.txt    3637_10.txt   5888_9.txt    8137_10.
txt
10388_7.txt   1388_10.txt   3638_10.txt   5889_7.txt    8138_10.
txt
10389_10.txt  1389_8.txt    3639_9.txt    588_9.txt     8139_10.
txt
1038_7.txt    138_7.txt     363_10.txt    5890_8.txt    813_10.t
xt
10390_10.txt  1390_9.txt    3640_8.txt    5891_10.txt   8140_10.
txt
10391_10.txt  1391_7.txt    3641_9.txt    5892_8.txt    8141_10.
```

```
txt
10392_10.txt  1392_7.txt    3642_9.txt    5893_9.txt    8142_10.
txt
10393_9.txt   1393_7.txt    3643_10.txt   5894_7.txt    8143_10.
txt
10394_10.txt  1394_8.txt    3644_8.txt    5895_10.txt   8144_9.t
xt
10395_8.txt   1395_10.txt   3645_8.txt    5896_10.txt   8145_10.
txt
10396_8.txt   1396_8.txt    3646_8.txt    5897_10.txt   8146_10.
txt
10397_8.txt   1397_7.txt    3647_10.txt   5898_8.txt    8147_10.
txt
10398_8.txt   1398_7.txt    3648_8.txt    5899_7.txt    8148_8.t
xt
10399_10.txt  1399_9.txt    3649_9.txt    589_10.txt    8149_10.
txt
1039_9.txt    139_10.txt    364_10.txt    58_9.txt      814_10.t
xt
103_7.txt     13_7.txt      3650_10.txt   5900_10.txt   8150_10.
txt
10400_10.txt  1400_7.txt    3651_10.txt   5901_7.txt    8151_10.
txt
10401_10.txt  1401_7.txt    3652_10.txt   5902_9.txt    8152_10.
txt
10402_10.txt  1402_7.txt    3653_10.txt   5903_8.txt    8153_9.t
xt
10403_7.txt   1403_7.txt    3654_10.txt   5904_8.txt    8154_10.
txt
10404_9.txt   1404_10.txt   3655_7.txt    5905_9.txt    8155_7.t
xt
10405_8.txt   1405_7.txt    3656_8.txt    5906_8.txt    8156_10.
txt
10406_10.txt  1406_8.txt    3657_7.txt    5907_10.txt   8157_10.
txt
10407_8.txt   1407_7.txt    3658_10.txt   5908_10.txt   8158_10.
txt
10408_10.txt  1408_7.txt    3659_9.txt    5909_8.txt    8159_10.
txt
10409_10.txt  1409_8.txt    365_10.txt    590_10.txt    815_7.tx
```

```
t
1040_10.txt   140_8.txt     3660_10.txt   5910_8.txt    8160_7.t
xt
10410_10.txt  1410_9.txt    3661_10.txt   5911_9.txt    8161_10.
txt
10411_9.txt   1411_7.txt    3662_9.txt    5912_7.txt    8162_9.t
xt
10412_8.txt   1412_8.txt    3663_10.txt   5913_10.txt   8163_8.t
xt
10413_10.txt  1413_7.txt    3664_10.txt   5914_8.txt    8164_8.t
xt
10414_10.txt  1414_10.txt   3665_9.txt    5915_7.txt    8165_8.t
xt
10415_7.txt   1415_10.txt   3666_9.txt    5916_10.txt   8166_10.
txt
10416_9.txt   1416_10.txt   3667_8.txt    5917_7.txt    8167_7.t
xt
10417_8.txt   1417_8.txt    3668_7.txt    5918_9.txt    8168_7.t
xt
10418_9.txt   1418_9.txt    3669_10.txt   5919_9.txt    8169_8.t
xt
10419_10.txt  1419_7.txt    366_9.txt     591_10.txt    816_10.t
xt
1041_9.txt    141_9.txt     3670_10.txt   5920_9.txt    8170_7.t
xt
10420_10.txt  1420_8.txt    3671_7.txt    5921_7.txt    8171_8.t
xt
10421_7.txt   1421_9.txt    3672_10.txt   5922_8.txt    8172_9.t
xt
10422_7.txt   1422_10.txt   3673_8.txt    5923_7.txt    8173_8.t
xt
10423_9.txt   1423_10.txt   3674_8.txt    5924_8.txt    8174_9.t
xt
10424_9.txt   1424_10.txt   3675_9.txt    5925_7.txt    8175_7.t
xt
10425_9.txt   1425_7.txt    3676_8.txt    5926_7.txt    8176_8.t
xt
10426_9.txt   1426_8.txt    3677_8.txt    5927_9.txt    8177_8.t
xt
10427_8.txt   1427_9.txt    3678_8.txt    5928_8.txt    8178_8.t
```

```
xt
10428_10.txt  1428_7.txt    3679_7.txt    5929_10.txt   8179_7.t
xt
10429_10.txt  1429_9.txt    367_10.txt    592_10.txt    817_10.t
xt
1042_10.txt   142_8.txt     3680_9.txt    5930_7.txt    8180_8.t
xt
10430_9.txt   1430_9.txt    3681_8.txt    5931_7.txt    8181_10.
txt
10431_10.txt  1431_10.txt   3682_8.txt    5932_10.txt   8182_7.t
xt
10432_10.txt  1432_7.txt    3683_9.txt    5933_7.txt    8183_7.t
xt
10433_9.txt   1433_10.txt   3684_8.txt    5934_10.txt   8184_7.t
xt
10434_10.txt  1434_10.txt   3685_8.txt    5935_9.txt    8185_8.t
xt
10435_7.txt   1435_8.txt    3686_10.txt   5936_10.txt   8186_10.
txt
10436_8.txt   1436_10.txt   3687_7.txt    5937_9.txt    8187_9.t
xt
10437_7.txt   1437_8.txt    3688_8.txt    5938_10.txt   8188_8.t
xt
10438_9.txt   1438_7.txt    3689_8.txt    5939_7.txt    8189_10.
txt
10439_8.txt   1439_7.txt    368_10.txt    593_9.txt     818_10.t
xt
1043_10.txt   143_7.txt     3690_9.txt    5940_7.txt    8190_10.
txt
10440_9.txt   1440_7.txt    3691_8.txt    5941_8.txt    8191_8.t
xt
10441_10.txt  1441_9.txt    3692_10.txt   5942_7.txt    8192_10.
txt
10442_10.txt  1442_7.txt    3693_8.txt    5943_10.txt   8193_10.
txt
10443_9.txt   1443_8.txt    3694_8.txt    5944_9.txt    8194_9.t
xt
10444_9.txt   1444_7.txt    3695_8.txt    5945_10.txt   8195_10.
txt
10445_10.txt  1445_8.txt    3696_7.txt    5946_7.txt    8196_8.t
```

```
xt
10446_10.txt  1446_10.txt   3697_7.txt    5947_7.txt    8197_8.t
xt
10447_10.txt  1447_8.txt    3698_10.txt   5948_8.txt    8198_7.t
xt
10448_10.txt  1448_8.txt    3699_10.txt   5949_10.txt   8199_10.
txt
10449_9.txt   1449_9.txt    369_10.txt    594_9.txt     819_10.t
xt
1044_8.txt    144_8.txt     36_10.txt     5950_10.txt   81_10.tx
t
10450_10.txt  1450_8.txt    3700_9.txt    5951_10.txt   8200_8.t
xt
10451_10.txt  1451_8.txt    3701_10.txt   5952_9.txt    8201_8.t
xt
10452_10.txt  1452_8.txt    3702_9.txt    5953_9.txt    8202_10.
txt
10453_10.txt  1453_8.txt    3703_9.txt    5954_7.txt    8203_7.t
xt
10454_9.txt   1454_7.txt    3704_8.txt    5955_10.txt   8204_8.t
xt
10455_10.txt  1455_7.txt    3705_10.txt   5956_9.txt    8205_8.t
xt
10456_10.txt  1456_9.txt    3706_7.txt    5957_9.txt    8206_10.
txt
10457_8.txt   1457_10.txt   3707_7.txt    5958_9.txt    8207_10.
txt
10458_10.txt  1458_9.txt    3708_7.txt    5959_7.txt    8208_8.t
xt
10459_9.txt   1459_10.txt   3709_7.txt    595_9.txt     8209_8.t
xt
1045_8.txt    145_10.txt    370_10.txt    5960_8.txt    820_10.t
xt
10460_10.txt  1460_10.txt   3710_9.txt    5961_10.txt   8210_7.t
xt
10461_9.txt   1461_10.txt   3711_7.txt    5962_7.txt    8211_7.t
xt
10462_7.txt   1462_9.txt    3712_7.txt    5963_10.txt   8212_7.t
xt
10463_10.txt  1463_10.txt   3713_10.txt   5964_8.txt    8213_9.t
```

```
xt
10464_7.txt    1464_7.txt     3714_10.txt    5965_10.txt    8214_7.t
xt
10465_8.txt    1465_9.txt     3715_7.txt     5966_7.txt     8215_7.t
xt
10466_8.txt    1466_10.txt    3716_9.txt     5967_10.txt    8216_8.t
xt
10467_10.txt   1467_9.txt     3717_8.txt     5968_10.txt    8217_8.t
xt
10468_9.txt    1468_9.txt     3718_8.txt     5969_10.txt    8218_7.t
xt
10469_10.txt   1469_7.txt     3719_9.txt     596_7.txt      8219_7.t
xt
1046_10.txt    146_10.txt     371_9.txt      5970_10.txt    821_10.t
xt
10470_9.txt    1470_10.txt    3720_10.txt    5971_7.txt     8220_8.t
xt
10471_10.txt   1471_9.txt     3721_8.txt     5972_8.txt     8221_9.t
xt
10472_7.txt    1472_8.txt     3722_10.txt    5973_9.txt     8222_9.t
xt
10473_10.txt   1473_9.txt     3723_8.txt     5974_7.txt     8223_9.t
xt
10474_9.txt    1474_8.txt     3724_9.txt     5975_8.txt     8224_8.t
xt
10475_8.txt    1475_8.txt     3725_8.txt     5976_7.txt     8225_8.t
xt
10476_9.txt    1476_10.txt    3726_7.txt     5977_8.txt     8226_8.t
xt
10477_9.txt    1477_7.txt     3727_10.txt    5978_9.txt     8227_7.t
xt
10478_8.txt    1478_7.txt     3728_10.txt    5979_9.txt     8228_8.t
xt
10479_10.txt   1479_8.txt     3729_10.txt    597_7.txt      8229_10.
txt
1047_8.txt     147_9.txt      372_10.txt     5980_10.txt    822_9.tx
t
10480_10.txt   1480_8.txt     3730_10.txt    5981_7.txt     8230_10.
txt
10481_8.txt    1481_10.txt    3731_10.txt    5982_9.txt     8231_10.
```

```
txt
10482_10.txt  1482_10.txt   3732_10.txt   5983_8.txt    8232_10.
txt
10483_8.txt   1483_9.txt    3733_10.txt   5984_10.txt   8233_8.t
xt
10484_8.txt   1484_10.txt   3734_9.txt    5985_10.txt   8234_7.t
xt
10485_8.txt   1485_10.txt   3735_10.txt   5986_10.txt   8235_7.t
xt
10486_7.txt   1486_7.txt    3736_8.txt    5987_8.txt    8236_7.t
xt
10487_7.txt   1487_9.txt    3737_9.txt    5988_7.txt    8237_8.t
xt
10488_10.txt  1488_10.txt   3738_8.txt    5989_7.txt    8238_8.t
xt
10489_10.txt  1489_8.txt    3739_10.txt   598_9.txt     8239_10.
txt
1048_8.txt    148_9.txt     373_10.txt    5990_7.txt    823_9.tx
t
10490_7.txt   1490_8.txt    3740_9.txt    5991_8.txt    8240_9.t
xt
10491_7.txt   1491_9.txt    3741_8.txt    5992_7.txt    8241_7.t
xt
10492_10.txt  1492_10.txt   3742_9.txt    5993_8.txt    8242_8.t
xt
10493_9.txt   1493_9.txt    3743_10.txt   5994_8.txt    8243_9.t
xt
10494_10.txt  1494_8.txt    3744_10.txt   5995_10.txt   8244_9.t
xt
10495_7.txt   1495_9.txt    3745_10.txt   5996_7.txt    8245_9.t
xt
10496_10.txt  1496_10.txt   3746_10.txt   5997_9.txt    8246_9.t
xt
10497_8.txt   1497_8.txt    3747_10.txt   5998_10.txt   8247_9.t
xt
10498_10.txt  1498_8.txt    3748_10.txt   5999_7.txt    8248_9.t
xt
10499_10.txt  1499_7.txt    3749_9.txt    599_10.txt    8249_10.
txt
1049_7.txt    149_10.txt    374_10.txt    59_7.txt      824_8.tx
```

```
t
104_10.txt    14_10.txt     3750_10.txt   5_10.txt      8250_8.t
xt
10500_10.txt  1500_9.txt    3751_10.txt   6000_9.txt    8251_10.
txt
10501_10.txt  1501_7.txt    3752_7.txt    6001_7.txt    8252_9.t
xt
10502_9.txt   1502_10.txt   3753_9.txt    6002_7.txt    8253_10.
txt
10503_10.txt  1503_9.txt    3754_8.txt    6003_8.txt    8254_8.t
xt
10504_9.txt   1504_9.txt    3755_10.txt   6004_10.txt   8255_9.t
xt
10505_10.txt  1505_9.txt    3756_8.txt    6005_8.txt    8256_9.t
xt
10506_10.txt  1506_7.txt    3757_7.txt    6006_10.txt   8257_9.t
xt
10507_10.txt  1507_8.txt    3758_9.txt    6007_8.txt    8258_9.t
xt
10508_10.txt  1508_7.txt    3759_8.txt    6008_7.txt    8259_9.t
xt
10509_7.txt   1509_10.txt   375_9.txt     6009_10.txt   825_10.t
xt
1050_9.txt    150_8.txt     3760_7.txt    600_10.txt    8260_9.t
xt
10510_7.txt   1510_8.txt    3761_8.txt    6010_8.txt    8261_8.t
xt
10511_7.txt   1511_8.txt    3762_8.txt    6011_8.txt    8262_10.
txt
10512_10.txt  1512_10.txt   3763_8.txt    6012_9.txt    8263_9.t
xt
10513_7.txt   1513_7.txt    3764_8.txt    6013_7.txt    8264_9.t
xt
10514_8.txt   1514_7.txt    3765_10.txt   6014_8.txt    8265_10.
txt
10515_9.txt   1515_7.txt    3766_10.txt   6015_7.txt    8266_10.
txt
10516_7.txt   1516_7.txt    3767_9.txt    6016_10.txt   8267_7.t
xt
10517_8.txt   1517_7.txt    3768_10.txt   6017_8.txt    8268_7.t
```

```
xt
10518_9.txt   1518_8.txt    3769_7.txt    6018_9.txt    8269_7.t
xt
10519_9.txt   1519_10.txt   376_10.txt    6019_8.txt    826_9.tx
t
1051_9.txt    151_10.txt    3770_10.txt   601_7.txt     8270_10.
txt
10520_9.txt   1520_7.txt    3771_10.txt   6020_7.txt    8271_10.
txt
10521_9.txt   1521_8.txt    3772_10.txt   6021_7.txt    8272_7.t
xt
10522_7.txt   1522_8.txt    3773_7.txt    6022_7.txt    8273_10.
txt
10523_9.txt   1523_9.txt    3774_8.txt    6023_8.txt    8274_10.
txt
10524_10.txt  1524_7.txt    3775_7.txt    6024_10.txt   8275_10.
txt
10525_10.txt  1525_9.txt    3776_10.txt   6025_9.txt    8276_10.
txt
10526_9.txt   1526_9.txt    3777_10.txt   6026_8.txt    8277_10.
txt
10527_10.txt  1527_7.txt    3778_10.txt   6027_9.txt    8278_10.
txt
10528_10.txt  1528_8.txt    3779_8.txt    6028_9.txt    8279_10.
txt
10529_10.txt  1529_10.txt   377_7.txt     6029_7.txt    827_7.tx
t
1052_8.txt    152_9.txt     3780_8.txt    602_10.txt    8280_8.t
xt
10530_10.txt  1530_10.txt   3781_7.txt    6030_10.txt   8281_7.t
xt
10531_10.txt  1531_8.txt    3782_8.txt    6031_7.txt    8282_7.t
xt
10532_8.txt   1532_10.txt   3783_10.txt   6032_10.txt   8283_7.t
xt
10533_10.txt  1533_8.txt    3784_9.txt    6033_8.txt    8284_8.t
xt
10534_7.txt   1534_10.txt   3785_10.txt   6034_10.txt   8285_8.t
xt
10535_7.txt   1535_9.txt    3786_9.txt    6035_10.txt   8286_7.t
```

```
xt
10536_10.txt  1536_7.txt    3787_10.txt   6036_10.txt   8287_9.t
xt
10537_10.txt  1537_9.txt    3788_10.txt   6037_10.txt   8288_7.t
xt
10538_8.txt   1538_10.txt   3789_8.txt    6038_10.txt   8289_8.t
xt
10539_10.txt  1539_10.txt   378_8.txt     6039_10.txt   828_10.t
xt
1053_8.txt    153_10.txt    3790_10.txt   603_10.txt    8290_10.
txt
10540_10.txt  1540_10.txt   3791_10.txt   6040_10.txt   8291_8.t
xt
10541_10.txt  1541_8.txt    3792_8.txt    6041_10.txt   8292_8.t
xt
10542_7.txt   1542_10.txt   3793_7.txt    6042_9.txt    8293_8.t
xt
10543_8.txt   1543_8.txt    3794_10.txt   6043_10.txt   8294_8.t
xt
10544_8.txt   1544_10.txt   3795_9.txt    6044_10.txt   8295_7.t
xt
10545_7.txt   1545_10.txt   3796_8.txt    6045_9.txt    8296_8.t
xt
10546_9.txt   1546_8.txt    3797_9.txt    6046_7.txt    8297_10.
txt
10547_9.txt   1547_7.txt    3798_9.txt    6047_10.txt   8298_7.t
xt
10548_7.txt   1548_8.txt    3799_8.txt    6048_9.txt    8299_7.t
xt
10549_9.txt   1549_10.txt   379_10.txt    6049_8.txt    829_7.tx
t
1054_8.txt    154_8.txt     37_9.txt      604_8.txt     82_8.txt
10550_8.txt   1550_10.txt   3800_7.txt    6050_10.txt   8300_10.
txt
10551_7.txt   1551_8.txt    3801_8.txt    6051_9.txt    8301_7.t
xt
10552_9.txt   1552_8.txt    3802_10.txt   6052_9.txt    8302_8.t
xt
10553_8.txt   1553_10.txt   3803_9.txt    6053_10.txt   8303_10.
txt
```

```
10554_7.txt    1554_8.txt     3804_10.txt    6054_9.txt     8304_8.t
xt
10555_8.txt    1555_8.txt     3805_8.txt     6055_8.txt     8305_7.t
xt
10556_10.txt   1556_8.txt     3806_8.txt     6056_9.txt     8306_10.
txt
10557_9.txt    1557_10.txt    3807_8.txt     6057_9.txt     8307_10.
txt
10558_10.txt   1558_10.txt    3808_10.txt    6058_8.txt     8308_7.t
xt
10559_8.txt    1559_7.txt     3809_9.txt     6059_10.txt    8309_7.t
xt
1055_10.txt    155_10.txt     380_10.txt     605_8.txt      830_10.t
xt
10560_9.txt    1560_9.txt     3810_9.txt     6060_10.txt    8310_10.
txt
10561_8.txt    1561_8.txt     3811_9.txt     6061_10.txt    8311_9.t
xt
10562_9.txt    1562_10.txt    3812_9.txt     6062_10.txt    8312_10.
txt
10563_7.txt    1563_10.txt    3813_7.txt     6063_10.txt    8313_10.
txt
10564_10.txt   1564_10.txt    3814_7.txt     6064_7.txt     8314_9.t
xt
10565_9.txt    1565_7.txt     3815_7.txt     6065_10.txt    8315_10.
txt
10566_8.txt    1566_8.txt     3816_7.txt     6066_10.txt    8316_9.t
xt
10567_9.txt    1567_7.txt     3817_8.txt     6067_10.txt    8317_8.t
xt
10568_10.txt   1568_9.txt     3818_8.txt     6068_9.txt     8318_10.
txt
10569_10.txt   1569_10.txt    3819_8.txt     6069_8.txt     8319_10.
txt
1056_10.txt    156_8.txt      381_10.txt     606_10.txt     831_9.tx
t
10570_8.txt    1570_8.txt     3820_8.txt     6070_8.txt     8320_8.t
xt
10571_8.txt    1571_10.txt    3821_9.txt     6071_8.txt     8321_7.t
xt
```

```
10572_8.txt   1572_10.txt   3822_10.txt   6072_7.txt    8322_7.t
xt
10573_10.txt  1573_9.txt    3823_9.txt    6073_9.txt    8323_10.
txt
10574_10.txt  1574_9.txt    3824_9.txt    6074_7.txt    8324_8.t
xt
10575_9.txt   1575_9.txt    3825_8.txt    6075_10.txt   8325_9.t
xt
10576_7.txt   1576_10.txt   3826_10.txt   6076_9.txt    8326_7.t
xt
10577_10.txt  1577_9.txt    3827_10.txt   6077_9.txt    8327_9.t
xt
10578_7.txt   1578_8.txt    3828_10.txt   6078_8.txt    8328_10.
txt
10579_10.txt  1579_10.txt   3829_10.txt   6079_9.txt    8329_7.t
xt
1057_9.txt    157_9.txt     382_10.txt    607_10.txt    832_8.tx
t
10580_8.txt   1580_7.txt    3830_10.txt   6080_10.txt   8330_7.t
xt
10581_10.txt  1581_7.txt    3831_10.txt   6081_9.txt    8331_8.t
xt
10582_10.txt  1582_8.txt    3832_8.txt    6082_9.txt    8332_9.t
xt
10583_10.txt  1583_8.txt    3833_8.txt    6083_8.txt    8333_9.t
xt
10584_10.txt  1584_9.txt    3834_9.txt    6084_8.txt    8334_8.t
xt
10585_9.txt   1585_9.txt    3835_9.txt    6085_8.txt    8335_8.t
xt
10586_10.txt  1586_7.txt    3836_8.txt    6086_8.txt    8336_10.
txt
10587_8.txt   1587_10.txt   3837_9.txt    6087_9.txt    8337_10.
txt
10588_10.txt  1588_8.txt    3838_7.txt    6088_10.txt   8338_8.t
xt
10589_10.txt  1589_10.txt   3839_7.txt    6089_10.txt   8339_10.
txt
1058_10.txt   158_10.txt    383_10.txt    608_8.txt     833_8.tx
t
```

```
10590_8.txt   1590_10.txt   3840_10.txt   6090_7.txt    8340_8.t
xt
10591_10.txt  1591_10.txt   3841_9.txt    6091_7.txt    8341_10.
txt
10592_8.txt   1592_9.txt    3842_7.txt    6092_8.txt    8342_7.t
xt
10593_8.txt   1593_10.txt   3843_7.txt    6093_8.txt    8343_7.t
xt
10594_8.txt   1594_8.txt    3844_8.txt    6094_10.txt   8344_10.
txt
10595_10.txt  1595_7.txt    3845_7.txt    6095_7.txt    8345_7.t
xt
10596_8.txt   1596_8.txt    3846_10.txt   6096_10.txt   8346_9.t
xt
10597_9.txt   1597_8.txt    3847_9.txt    6097_10.txt   8347_10.
txt
10598_8.txt   1598_8.txt    3848_8.txt    6098_10.txt   8348_10.
txt
10599_8.txt   1599_7.txt    3849_8.txt    6099_8.txt    8349_10.
txt
1059_10.txt   159_10.txt    384_8.txt     609_9.txt     834_8.tx
t
105_7.txt     15_7.txt      3850_8.txt    60_8.txt      8350_9.t
xt
10600_9.txt   1600_7.txt    3851_10.txt   6100_10.txt   8351_9.t
xt
10601_10.txt  1601_8.txt    3852_8.txt    6101_8.txt    8352_9.t
xt
10602_10.txt  1602_9.txt    3853_9.txt    6102_10.txt   8353_10.
txt
10603_10.txt  1603_9.txt    3854_10.txt   6103_10.txt   8354_8.t
xt
10604_7.txt   1604_9.txt    3855_8.txt    6104_10.txt   8355_7.t
xt
10605_7.txt   1605_10.txt   3856_10.txt   6105_10.txt   8356_10.
txt
10606_10.txt  1606_7.txt    3857_9.txt    6106_9.txt    8357_10.
txt
10607_10.txt  1607_9.txt    3858_10.txt   6107_7.txt    8358_9.t
xt
```

```
10608_10.txt  1608_9.txt    3859_8.txt    6108_8.txt    8359_8.t
xt
10609_10.txt  1609_9.txt    385_10.txt    6109_8.txt    835_8.tx
t
1060_10.txt   160_9.txt     3860_10.txt   610_9.txt     8360_9.t
xt
10610_8.txt   1610_10.txt   3861_10.txt   6110_8.txt    8361_8.t
xt
10611_8.txt   1611_10.txt   3862_9.txt    6111_10.txt   8362_7.t
xt
10612_7.txt   1612_10.txt   3863_7.txt    6112_7.txt    8363_8.t
xt
10613_8.txt   1613_10.txt   3864_7.txt    6113_8.txt    8364_10.
txt
10614_7.txt   1614_10.txt   3865_8.txt    6114_7.txt    8365_8.t
xt
10615_8.txt   1615_8.txt    3866_7.txt    6115_10.txt   8366_10.
txt
10616_7.txt   1616_9.txt    3867_10.txt   6116_8.txt    8367_10.
txt
10617_8.txt   1617_8.txt    3868_10.txt   6117_10.txt   8368_10.
txt
10618_8.txt   1618_10.txt   3869_9.txt    6118_10.txt   8369_8.t
xt
10619_8.txt   1619_10.txt   386_7.txt     6119_9.txt    836_8.tx
t
1061_10.txt   161_8.txt     3870_9.txt    611_10.txt    8370_10.
txt
10620_10.txt  1620_10.txt   3871_8.txt    6120_9.txt    8371_10.
txt
10621_10.txt  1621_10.txt   3872_9.txt    6121_10.txt   8372_10.
txt
10622_10.txt  1622_8.txt    3873_9.txt    6122_8.txt    8373_9.t
xt
10623_8.txt   1623_10.txt   3874_9.txt    6123_7.txt    8374_7.t
xt
10624_7.txt   1624_10.txt   3875_9.txt    6124_7.txt    8375_8.t
xt
10625_7.txt   1625_7.txt    3876_8.txt    6125_9.txt    8376_10.
txt
```

```
10626_7.txt    1626_10.txt    3877_8.txt     6126_10.txt    8377_8.t
xt
10627_10.txt   1627_10.txt    3878_10.txt    6127_8.txt     8378_10.
txt
10628_7.txt    1628_10.txt    3879_8.txt     6128_7.txt     8379_10.
txt
10629_10.txt   1629_10.txt    387_8.txt      6129_7.txt     837_7.tx
t
1062_10.txt    162_8.txt      3880_8.txt     612_10.txt     8380_9.t
xt
10630_8.txt    1630_8.txt     3881_9.txt     6130_7.txt     8381_8.t
xt
10631_8.txt    1631_8.txt     3882_10.txt    6131_10.txt    8382_7.t
xt
10632_10.txt   1632_7.txt     3883_10.txt    6132_10.txt    8383_7.t
xt
10633_9.txt    1633_9.txt     3884_8.txt     6133_7.txt     8384_7.t
xt
10634_10.txt   1634_7.txt     3885_10.txt    6134_7.txt     8385_10.
txt
10635_8.txt    1635_7.txt     3886_10.txt    6135_8.txt     8386_9.t
xt
10636_8.txt    1636_10.txt    3887_10.txt    6136_10.txt    8387_8.t
xt
10637_10.txt   1637_10.txt    3888_10.txt    6137_7.txt     8388_7.t
xt
10638_8.txt    1638_8.txt     3889_10.txt    6138_8.txt     8389_8.t
xt
10639_7.txt    1639_10.txt    388_8.txt      6139_10.txt    838_9.tx
t
1063_10.txt    163_10.txt     3890_10.txt    613_10.txt     8390_8.t
xt
10640_8.txt    1640_10.txt    3891_10.txt    6140_9.txt     8391_8.t
xt
10641_7.txt    1641_10.txt    3892_10.txt    6141_9.txt     8392_8.t
xt
10642_8.txt    1642_10.txt    3893_10.txt    6142_9.txt     8393_8.t
xt
10643_8.txt    1643_10.txt    3894_10.txt    6143_7.txt     8394_10.
txt
```

```
10644_8.txt    1644_10.txt    3895_8.txt     6144_7.txt     8395_8.t
xt
10645_8.txt    1645_9.txt     3896_7.txt     6145_8.txt     8396_8.t
xt
10646_8.txt    1646_9.txt     3897_10.txt    6146_7.txt     8397_10.
txt
10647_8.txt    1647_10.txt    3898_10.txt    6147_7.txt     8398_8.t
xt
10648_8.txt    1648_10.txt    3899_9.txt     6148_10.txt    8399_10.
txt
10649_7.txt    1649_8.txt     389_10.txt     6149_7.txt     839_7.tx
t
1064_10.txt    164_10.txt     38_10.txt      614_10.txt     83_10.tx
t
10650_8.txt    1650_10.txt    3900_10.txt    6150_7.txt     8400_7.t
xt
10651_7.txt    1651_10.txt    3901_10.txt    6151_7.txt     8401_10.
txt
10652_9.txt    1652_10.txt    3902_10.txt    6152_10.txt    8402_10.
txt
10653_10.txt   1653_10.txt    3903_10.txt    6153_7.txt     8403_10.
txt
10654_7.txt    1654_7.txt     3904_10.txt    6154_10.txt    8404_10.
txt
10655_9.txt    1655_8.txt     3905_10.txt    6155_7.txt     8405_9.t
xt
10656_7.txt    1656_9.txt     3906_10.txt    6156_10.txt    8406_10.
txt
10657_8.txt    1657_9.txt     3907_10.txt    6157_10.txt    8407_7.t
xt
10658_10.txt   1658_10.txt    3908_10.txt    6158_10.txt    8408_10.
txt
10659_8.txt    1659_7.txt     3909_10.txt    6159_7.txt     8409_8.t
xt
1065_10.txt    165_7.txt      390_10.txt     615_10.txt     840_9.tx
t
10660_10.txt   1660_8.txt     3910_10.txt    6160_10.txt    8410_10.
txt
10661_9.txt    1661_8.txt     3911_10.txt    6161_9.txt     8411_7.t
xt
```

```
10662_7.txt    1662_7.txt     3912_10.txt    6162_8.txt     8412_7.t
xt
10663_8.txt    1663_9.txt     3913_10.txt    6163_10.txt    8413_8.t
xt
10664_8.txt    1664_10.txt    3914_10.txt    6164_7.txt     8414_7.t
xt
10665_8.txt    1665_7.txt     3915_9.txt     6165_8.txt     8415_7.t
xt
10666_8.txt    1666_8.txt     3916_8.txt     6166_10.txt    8416_7.t
xt
10667_8.txt    1667_7.txt     3917_9.txt     6167_7.txt     8417_8.t
xt
10668_7.txt    1668_8.txt     3918_10.txt    6168_10.txt    8418_7.t
xt
10669_10.txt   1669_8.txt     3919_7.txt     6169_10.txt    8419_9.t
xt
1066_10.txt    166_7.txt      391_8.txt      616_7.txt      841_10.t
xt
10670_10.txt   1670_8.txt     3920_9.txt     6170_10.txt    8420_9.t
xt
10671_10.txt   1671_8.txt     3921_9.txt     6171_8.txt     8421_10.
txt
10672_9.txt    1672_8.txt     3922_10.txt    6172_7.txt     8422_10.
txt
10673_10.txt   1673_8.txt     3923_10.txt    6173_8.txt     8423_8.t
xt
10674_8.txt    1674_8.txt     3924_7.txt     6174_10.txt    8424_9.t
xt
10675_8.txt    1675_9.txt     3925_10.txt    6175_8.txt     8425_9.t
xt
10676_9.txt    1676_9.txt     3926_7.txt     6176_9.txt     8426_7.t
xt
10677_8.txt    1677_9.txt     3927_9.txt     6177_7.txt     8427_7.t
xt
10678_9.txt    1678_9.txt     3928_7.txt     6178_7.txt     8428_7.t
xt
10679_10.txt   1679_9.txt     3929_9.txt     6179_8.txt     8429_7.t
xt
1067_7.txt     167_7.txt      392_9.txt      617_7.txt      842_9.tx
t
```

```
10680_8.txt    1680_8.txt     3930_9.txt     6180_7.txt     8430_9.t
xt
10681_10.txt   1681_7.txt     3931_7.txt     6181_9.txt     8431_9.t
xt
10682_10.txt   1682_7.txt     3932_8.txt     6182_7.txt     8432_10.
txt
10683_7.txt    1683_7.txt     3933_7.txt     6183_7.txt     8433_9.t
xt
10684_9.txt    1684_10.txt    3934_8.txt     6184_7.txt     8434_9.t
xt
10685_7.txt    1685_10.txt    3935_9.txt     6185_8.txt     8435_10.
txt
10686_8.txt    1686_10.txt    3936_7.txt     6186_10.txt    8436_10.
txt
10687_10.txt   1687_10.txt    3937_8.txt     6187_9.txt     8437_10.
txt
10688_9.txt    1688_9.txt     3938_9.txt     6188_9.txt     8438_10.
txt
10689_8.txt    1689_10.txt    3939_7.txt     6189_10.txt    8439_10.
txt
1068_10.txt    168_9.txt      393_8.txt      618_10.txt     843_10.t
xt
10690_10.txt   1690_10.txt    3940_9.txt     6190_7.txt     8440_8.t
xt
10691_7.txt    1691_8.txt     3941_9.txt     6191_9.txt     8441_10.
txt
10692_8.txt    1692_8.txt     3942_9.txt     6192_9.txt     8442_9.t
xt
10693_8.txt    1693_10.txt    3943_10.txt    6193_7.txt     8443_9.t
xt
10694_7.txt    1694_10.txt    3944_8.txt     6194_8.txt     8444_10.
txt
10695_8.txt    1695_10.txt    3945_10.txt    6195_10.txt    8445_10.
txt
10696_7.txt    1696_10.txt    3946_7.txt     6196_8.txt     8446_10.
txt
10697_8.txt    1697_10.txt    3947_7.txt     6197_10.txt    8447_8.t
xt
10698_9.txt    1698_10.txt    3948_9.txt     6198_7.txt     8448_10.
txt
```

```
10699_9.txt    1699_10.txt    3949_8.txt     6199_8.txt     8449_9.t
xt
1069_10.txt    169_8.txt      394_8.txt      619_9.txt      844_8.tx
t
106_10.txt     16_7.txt       3950_7.txt     61_10.txt      8450_7.t
xt
10700_8.txt    1700_8.txt     3951_10.txt    6200_7.txt     8451_10.
txt
10701_10.txt   1701_10.txt    3952_10.txt    6201_9.txt     8452_7.t
xt
10702_10.txt   1702_9.txt     3953_10.txt    6202_7.txt     8453_10.
txt
10703_7.txt    1703_8.txt     3954_9.txt     6203_7.txt     8454_8.t
xt
10704_10.txt   1704_8.txt     3955_9.txt     6204_7.txt     8455_10.
txt
10705_7.txt    1705_10.txt    3956_10.txt    6205_8.txt     8456_10.
txt
10706_7.txt    1706_9.txt     3957_10.txt    6206_8.txt     8457_9.t
xt
10707_8.txt    1707_10.txt    3958_7.txt     6207_9.txt     8458_10.
txt
10708_8.txt    1708_10.txt    3959_9.txt     6208_7.txt     8459_10.
txt
10709_10.txt   1709_8.txt     395_10.txt     6209_8.txt     845_7.tx
t
1070_8.txt     170_10.txt     3960_10.txt    620_10.txt     8460_7.t
xt
10710_9.txt    1710_7.txt     3961_8.txt     6210_10.txt    8461_7.t
xt
10711_10.txt   1711_8.txt     3962_10.txt    6211_8.txt     8462_9.t
xt
10712_8.txt    1712_9.txt     3963_7.txt     6212_9.txt     8463_9.t
xt
10713_9.txt    1713_8.txt     3964_7.txt     6213_7.txt     8464_10.
txt
10714_8.txt    1714_8.txt     3965_8.txt     6214_7.txt     8465_8.t
xt
10715_8.txt    1715_8.txt     3966_9.txt     6215_7.txt     8466_9.t
xt
```

```
10716_7.txt   1716_8.txt    3967_7.txt    6216_7.txt    8467_7.t
xt
10717_10.txt  1717_8.txt    3968_10.txt   6217_8.txt    8468_7.t
xt
10718_10.txt  1718_7.txt    3969_8.txt    6218_7.txt    8469_7.t
xt
10719_10.txt  1719_7.txt    396_8.txt     6219_7.txt    846_7.tx
t
1071_8.txt    171_8.txt     3970_7.txt    621_10.txt    8470_8.t
xt
10720_9.txt   1720_10.txt   3971_8.txt    6220_8.txt    8471_10.
txt
10721_9.txt   1721_8.txt    3972_7.txt    6221_7.txt    8472_10.
txt
10722_10.txt  1722_7.txt    3973_7.txt    6222_9.txt    8473_8.t
xt
10723_8.txt   1723_7.txt    3974_8.txt    6223_7.txt    8474_10.
txt
10724_8.txt   1724_10.txt   3975_10.txt   6224_7.txt    8475_10.
txt
10725_9.txt   1725_8.txt    3976_10.txt   6225_8.txt    8476_10.
txt
10726_7.txt   1726_8.txt    3977_10.txt   6226_10.txt   8477_10.
txt
10727_7.txt   1727_10.txt   3978_10.txt   6227_7.txt    8478_8.t
xt
10728_10.txt  1728_7.txt    3979_10.txt   6228_7.txt    8479_9.t
xt
10729_8.txt   1729_8.txt    397_9.txt     6229_7.txt    847_7.tx
t
1072_10.txt   172_10.txt    3980_10.txt   622_10.txt    8480_10.
txt
10730_10.txt  1730_10.txt   3981_10.txt   6230_8.txt    8481_8.t
xt
10731_7.txt   1731_10.txt   3982_7.txt    6231_10.txt   8482_7.t
xt
10732_8.txt   1732_10.txt   3983_10.txt   6232_10.txt   8483_7.t
xt
10733_7.txt   1733_7.txt    3984_7.txt    6233_9.txt    8484_10.
txt
```

```
10734_10.txt  1734_7.txt    3985_8.txt    6234_8.txt    8485_8.t
xt
10735_10.txt  1735_10.txt   3986_10.txt   6235_8.txt    8486_10.
txt
10736_10.txt  1736_10.txt   3987_7.txt    6236_8.txt    8487_10.
txt
10737_10.txt  1737_8.txt    3988_7.txt    6237_7.txt    8488_8.t
xt
10738_9.txt   1738_7.txt    3989_8.txt    6238_9.txt    8489_9.t
xt
10739_10.txt  1739_10.txt   398_10.txt    6239_8.txt    848_8.tx
t
1073_9.txt    173_7.txt     3990_8.txt    623_10.txt    8490_7.t
xt
10740_8.txt   1740_8.txt    3991_7.txt    6240_10.txt   8491_7.t
xt
10741_10.txt  1741_9.txt    3992_8.txt    6241_10.txt   8492_9.t
xt
10742_9.txt   1742_7.txt    3993_7.txt    6242_8.txt    8493_10.
txt
10743_9.txt   1743_9.txt    3994_8.txt    6243_9.txt    8494_8.t
xt
10744_8.txt   1744_10.txt   3995_8.txt    6244_8.txt    8495_8.t
xt
10745_10.txt  1745_7.txt    3996_9.txt    6245_10.txt   8496_7.t
xt
10746_10.txt  1746_7.txt    3997_10.txt   6246_9.txt    8497_7.t
xt
10747_10.txt  1747_10.txt   3998_10.txt   6247_10.txt   8498_9.t
xt
10748_10.txt  1748_8.txt    3999_10.txt   6248_7.txt    8499_9.t
xt
10749_8.txt   1749_10.txt   399_9.txt     6249_7.txt    849_7.tx
t
1074_10.txt   174_7.txt     39_9.txt      624_9.txt     84_10.tx
t
10750_8.txt   1750_10.txt   3_10.txt      6250_10.txt   8500_7.t
xt
10751_10.txt  1751_8.txt    4000_10.txt   6251_7.txt    8501_8.t
xt
```

```
10752_10.txt  1752_8.txt    4001_8.txt    6252_10.txt   8502_9.t
xt
10753_10.txt  1753_9.txt    4002_8.txt    6253_8.txt    8503_8.t
xt
10754_10.txt  1754_10.txt   4003_9.txt    6254_10.txt   8504_8.t
xt
10755_10.txt  1755_10.txt   4004_9.txt    6255_8.txt    8505_8.t
xt
10756_8.txt   1756_7.txt    4005_10.txt   6256_8.txt    8506_8.t
xt
10757_10.txt  1757_8.txt    4006_8.txt    6257_10.txt   8507_10.
txt
10758_8.txt   1758_10.txt   4007_9.txt    6258_8.txt    8508_7.t
xt
10759_9.txt   1759_8.txt    4008_9.txt    6259_8.txt    8509_7.t
xt
1075_10.txt   175_7.txt     4009_9.txt    625_10.txt    850_8.tx
t
10760_8.txt   1760_10.txt   400_10.txt    6260_10.txt   8510_7.t
xt
10761_10.txt  1761_9.txt    4010_10.txt   6261_8.txt    8511_10.
txt
10762_10.txt  1762_7.txt    4011_8.txt    6262_9.txt    8512_7.t
xt
10763_8.txt   1763_9.txt    4012_8.txt    6263_7.txt    8513_9.t
xt
10764_9.txt   1764_10.txt   4013_7.txt    6264_7.txt    8514_7.t
xt
10765_10.txt  1765_8.txt    4014_10.txt   6265_7.txt    8515_8.t
xt
10766_7.txt   1766_10.txt   4015_7.txt    6266_8.txt    8516_9.t
xt
10767_10.txt  1767_8.txt    4016_10.txt   6267_10.txt   8517_8.t
xt
10768_7.txt   1768_9.txt    4017_7.txt    6268_9.txt    8518_10.
txt
10769_10.txt  1769_8.txt    4018_7.txt    6269_10.txt   8519_10.
txt
1076_8.txt    176_7.txt     4019_7.txt    626_9.txt     851_7.tx
t
```

```
10770_7.txt   1770_10.txt   401_10.txt    6270_8.txt    8520_8.t
xt
10771_10.txt  1771_10.txt   4020_10.txt   6271_9.txt    8521_10.
txt
10772_10.txt  1772_10.txt   4021_7.txt    6272_7.txt    8522_10.
txt
10773_9.txt   1773_9.txt    4022_8.txt    6273_9.txt    8523_9.t
xt
10774_8.txt   1774_10.txt   4023_9.txt    6274_8.txt    8524_10.
txt
10775_8.txt   1775_9.txt    4024_9.txt    6275_7.txt    8525_10.
txt
10776_8.txt   1776_10.txt   4025_9.txt    6276_7.txt    8526_10.
txt
10777_9.txt   1777_10.txt   4026_9.txt    6277_7.txt    8527_8.t
xt
10778_8.txt   1778_10.txt   4027_10.txt   6278_7.txt    8528_7.t
xt
10779_10.txt  1779_10.txt   4028_10.txt   6279_10.txt   8529_9.t
xt
1077_8.txt    177_9.txt     4029_10.txt   627_8.txt     852_7.tx
t
10780_10.txt  1780_10.txt   402_10.txt    6280_7.txt    8530_9.t
xt
10781_10.txt  1781_10.txt   4030_9.txt    6281_8.txt    8531_7.t
xt
10782_7.txt   1782_7.txt    4031_10.txt   6282_7.txt    8532_7.t
xt
10783_10.txt  1783_8.txt    4032_9.txt    6283_7.txt    8533_8.t
xt
10784_10.txt  1784_10.txt   4033_9.txt    6284_7.txt    8534_10.
txt
10785_10.txt  1785_9.txt    4034_9.txt    6285_7.txt    8535_7.t
xt
10786_10.txt  1786_7.txt    4035_10.txt   6286_8.txt    8536_9.t
xt
10787_10.txt  1787_10.txt   4036_7.txt    6287_10.txt   8537_10.
txt
10788_10.txt  1788_7.txt    4037_7.txt    6288_7.txt    8538_10.
txt
```

```
10789_10.txt   1789_7.txt    4038_10.txt   6289_10.txt   8539_8.t
xt
1078_8.txt     178_7.txt     4039_10.txt   628_9.txt     853_9.tx
t
10790_8.txt    1790_9.txt    403_8.txt     6290_7.txt    8540_9.t
xt
10791_9.txt    1791_8.txt    4040_8.txt    6291_10.txt   8541_10.
txt
10792_9.txt    1792_10.txt   4041_7.txt    6292_10.txt   8542_9.t
xt
10793_10.txt   1793_9.txt    4042_7.txt    6293_10.txt   8543_8.t
xt
10794_10.txt   1794_7.txt    4043_7.txt    6294_10.txt   8544_9.t
xt
10795_7.txt    1795_8.txt    4044_9.txt    6295_7.txt    8545_9.t
xt
10796_9.txt    1796_8.txt    4045_10.txt   6296_7.txt    8546_10.
txt
10797_8.txt    1797_9.txt    4046_8.txt    6297_10.txt   8547_10.
txt
10798_8.txt    1798_9.txt    4047_10.txt   6298_9.txt    8548_10.
txt
10799_7.txt    1799_7.txt    4048_7.txt    6299_8.txt    8549_8.t
xt
1079_7.txt     179_8.txt     4049_7.txt    629_9.txt     854_9.tx
t
107_10.txt     17_9.txt      404_9.txt     62_10.txt     8550_8.t
xt
10800_8.txt    1800_8.txt    4050_9.txt    6300_8.txt    8551_8.t
xt
10801_8.txt    1801_8.txt    4051_8.txt    6301_10.txt   8552_9.t
xt
10802_8.txt    1802_9.txt    4052_8.txt    6302_8.txt    8553_7.t
xt
10803_8.txt    1803_10.txt   4053_8.txt    6303_8.txt    8554_7.t
xt
10804_10.txt   1804_10.txt   4054_9.txt    6304_8.txt    8555_9.t
xt
10805_10.txt   1805_10.txt   4055_10.txt   6305_10.txt   8556_7.t
xt
```

```
10806_9.txt    1806_8.txt     4056_7.txt     6306_10.txt    8557_7.t
xt
10807_9.txt    1807_7.txt     4057_7.txt     6307_8.txt     8558_10.
txt
10808_10.txt   1808_7.txt     4058_10.txt    6308_8.txt     8559_8.t
xt
10809_10.txt   1809_10.txt    4059_8.txt     6309_8.txt     855_9.tx
t
1080_9.txt     180_9.txt      405_10.txt     630_10.txt     8560_10.
txt
10810_8.txt    1810_7.txt     4060_10.txt    6310_10.txt    8561_10.
txt
10811_7.txt    1811_10.txt    4061_10.txt    6311_10.txt    8562_10.
txt
10812_8.txt    1812_10.txt    4062_10.txt    6312_7.txt     8563_10.
txt
10813_10.txt   1813_8.txt     4063_8.txt     6313_9.txt     8564_7.t
xt
10814_7.txt    1814_10.txt    4064_10.txt    6314_9.txt     8565_7.t
xt
10815_10.txt   1815_10.txt    4065_10.txt    6315_10.txt    8566_10.
txt
10816_10.txt   1816_9.txt     4066_10.txt    6316_8.txt     8567_10.
txt
10817_10.txt   1817_8.txt     4067_8.txt     6317_10.txt    8568_7.t
xt
10818_10.txt   1818_8.txt     4068_10.txt    6318_7.txt     8569_9.t
xt
10819_10.txt   1819_9.txt     4069_10.txt    6319_10.txt    856_7.tx
t
1081_10.txt    181_10.txt     406_8.txt      631_10.txt     8570_7.t
xt
10820_10.txt   1820_9.txt     4070_10.txt    6320_10.txt    8571_7.t
xt
10821_8.txt    1821_8.txt     4071_10.txt    6321_7.txt     8572_7.t
xt
10822_10.txt   1822_8.txt     4072_10.txt    6322_7.txt     8573_10.
txt
10823_8.txt    1823_7.txt     4073_10.txt    6323_7.txt     8574_9.t
xt
```

```
10824_10.txt  1824_8.txt   4074_10.txt  6324_8.txt   8575_10.
txt
10825_9.txt   1825_10.txt  4075_10.txt  6325_7.txt   8576_8.t
xt
10826_10.txt  1826_10.txt  4076_10.txt  6326_10.txt  8577_8.t
xt
10827_10.txt  1827_10.txt  4077_10.txt  6327_10.txt  8578_8.t
xt
10828_10.txt  1828_8.txt   4078_10.txt  6328_10.txt  8579_8.t
xt
10829_10.txt  1829_8.txt   4079_9.txt   6329_10.txt  857_8.tx
t
1082_10.txt   182_10.txt   407_10.txt   632_10.txt   8580_9.t
xt
10830_10.txt  1830_8.txt   4080_10.txt  6330_10.txt  8581_10.
txt
10831_7.txt   1831_10.txt  4081_10.txt  6331_10.txt  8582_9.t
xt
10832_10.txt  1832_7.txt   4082_10.txt  6332_8.txt   8583_9.t
xt
10833_10.txt  1833_10.txt  4083_10.txt  6333_8.txt   8584_8.t
xt
10834_7.txt   1834_8.txt   4084_7.txt   6334_8.txt   8585_8.t
xt
10835_10.txt  1835_10.txt  4085_10.txt  6335_10.txt  8586_8.t
xt
10836_10.txt  1836_7.txt   4086_7.txt   6336_7.txt   8587_7.t
xt
10837_10.txt  1837_10.txt  4087_10.txt  6337_7.txt   8588_7.t
xt
10838_10.txt  1838_10.txt  4088_7.txt   6338_9.txt   8589_7.t
xt
10839_10.txt  1839_10.txt  4089_8.txt   6339_7.txt   858_8.tx
t
1083_10.txt   183_8.txt    408_10.txt   633_8.txt    8590_7.t
xt
10840_9.txt   1840_10.txt  4090_8.txt   6340_10.txt  8591_7.t
xt
10841_10.txt  1841_7.txt   4091_7.txt   6341_7.txt   8592_7.t
xt
```

```
10842_7.txt    1842_9.txt     4092_10.txt    6342_10.txt    8593_7.t
xt
10843_7.txt    1843_9.txt     4093_8.txt     6343_10.txt    8594_7.t
xt
10844_9.txt    1844_8.txt     4094_9.txt     6344_7.txt     8595_10.
txt
10845_10.txt   1845_7.txt     4095_8.txt     6345_7.txt     8596_7.t
xt
10846_9.txt    1846_8.txt     4096_10.txt    6346_10.txt    8597_9.t
xt
10847_10.txt   1847_10.txt    4097_8.txt     6347_9.txt     8598_7.t
xt
10848_10.txt   1848_8.txt     4098_10.txt    6348_10.txt    8599_7.t
xt
10849_10.txt   1849_7.txt     4099_8.txt     6349_10.txt    859_9.tx
t
1084_9.txt     184_8.txt      409_10.txt     634_8.txt      85_10.tx
t
10850_10.txt   1850_10.txt    40_8.txt       6350_8.txt     8600_8.t
xt
10851_9.txt    1851_10.txt    4100_10.txt    6351_8.txt     8601_10.
txt
10852_10.txt   1852_9.txt     4101_8.txt     6352_10.txt    8602_10.
txt
10853_10.txt   1853_8.txt     4102_10.txt    6353_10.txt    8603_10.
txt
10854_10.txt   1854_10.txt    4103_7.txt     6354_10.txt    8604_10.
txt
10855_9.txt    1855_9.txt     4104_9.txt     6355_8.txt     8605_10.
txt
10856_8.txt    1856_9.txt     4105_10.txt    6356_8.txt     8606_10.
txt
10857_8.txt    1857_10.txt    4106_7.txt     6357_9.txt     8607_9.t
xt
10858_8.txt    1858_10.txt    4107_10.txt    6358_10.txt    8608_9.t
xt
10859_7.txt    1859_8.txt     4108_7.txt     6359_10.txt    8609_10.
txt
1085_7.txt     185_9.txt      4109_10.txt    635_9.txt      860_8.tx
t
```

```
10860_7.txt   1860_9.txt    410_8.txt     6360_7.txt    8610_10.txt
10861_7.txt   1861_10.txt   4110_10.txt   6361_10.txt   8611_8.txt
10862_9.txt   1862_8.txt    4111_10.txt   6362_7.txt    8612_7.txt
10863_8.txt   1863_10.txt   4112_9.txt    6363_8.txt    8613_7.txt
10864_8.txt   1864_10.txt   4113_7.txt    6364_8.txt    8614_9.txt
10865_7.txt   1865_8.txt    4114_9.txt    6365_7.txt    8615_10.txt
10866_7.txt   1866_8.txt    4115_8.txt    6366_8.txt    8616_8.txt
10867_7.txt   1867_9.txt    4116_9.txt    6367_7.txt    8617_9.txt
10868_8.txt   1868_10.txt   4117_9.txt    6368_7.txt    8618_8.txt
10869_7.txt   1869_9.txt    4118_10.txt   6369_10.txt   8619_7.txt
1086_7.txt    186_8.txt     4119_8.txt    636_10.txt    861_7.txt
10870_8.txt   1870_10.txt   411_10.txt    6370_8.txt    8620_8.txt
10871_7.txt   1871_10.txt   4120_9.txt    6371_9.txt    8621_10.txt
10872_7.txt   1872_7.txt    4121_10.txt   6372_7.txt    8622_8.txt
10873_8.txt   1873_8.txt    4122_10.txt   6373_10.txt   8623_7.txt
10874_10.txt  1874_10.txt   4123_7.txt    6374_10.txt   8624_7.txt
10875_8.txt   1875_10.txt   4124_8.txt    6375_8.txt    8625_7.txt
10876_7.txt   1876_10.txt   4125_8.txt    6376_10.txt   8626_7.txt
10877_10.txt  1877_7.txt    4126_8.txt    6377_9.txt    8627_10.txt
10878_7.txt   1878_10.txt   4127_8.txt    6378_8.txt    8628_10.txt
```

```
10879_10.txt  1879_10.txt   4128_10.txt   6379_9.txt    8629_9.t
xt
1087_10.txt   187_8.txt     4129_10.txt   637_10.txt    862_7.tx
t
10880_8.txt   1880_10.txt   412_8.txt     6380_9.txt    8630_9.t
xt
10881_7.txt   1881_7.txt    4130_9.txt    6381_8.txt    8631_7.t
xt
10882_8.txt   1882_10.txt   4131_7.txt    6382_7.txt    8632_9.t
xt
10883_7.txt   1883_7.txt    4132_7.txt    6383_10.txt   8633_8.t
xt
10884_8.txt   1884_8.txt    4133_7.txt    6384_9.txt    8634_9.t
xt
10885_7.txt   1885_10.txt   4134_7.txt    6385_10.txt   8635_7.t
xt
10886_10.txt  1886_10.txt   4135_10.txt   6386_10.txt   8636_7.t
xt
10887_7.txt   1887_8.txt    4136_10.txt   6387_8.txt    8637_10.
txt
10888_8.txt   1888_8.txt    4137_8.txt    6388_7.txt    8638_7.t
xt
10889_10.txt  1889_10.txt   4138_10.txt   6389_9.txt    8639_9.t
xt
1088_9.txt    188_7.txt     4139_8.txt    638_10.txt    863_10.t
xt
10890_9.txt   1890_10.txt   413_10.txt    6390_8.txt    8640_10.
txt
10891_7.txt   1891_8.txt    4140_10.txt   6391_8.txt    8641_7.t
xt
10892_7.txt   1892_8.txt    4141_10.txt   6392_10.txt   8642_8.t
xt
10893_8.txt   1893_10.txt   4142_10.txt   6393_8.txt    8643_7.t
xt
10894_8.txt   1894_8.txt    4143_9.txt    6394_10.txt   8644_7.t
xt
10895_7.txt   1895_10.txt   4144_10.txt   6395_9.txt    8645_8.t
xt
10896_8.txt   1896_8.txt    4145_10.txt   6396_9.txt    8646_7.t
xt
```

```
10897_9.txt    1897_10.txt    4146_10.txt    6397_8.txt     8647_7.t
xt
10898_7.txt    1898_9.txt     4147_8.txt     6398_8.txt     8648_9.t
xt
10899_10.txt   1899_7.txt     4148_7.txt     6399_7.txt     8649_9.t
xt
1089_10.txt    189_9.txt      4149_10.txt    639_10.txt     864_9.tx
t
108_10.txt     18_7.txt       414_10.txt     63_10.txt      8650_10.
txt
10900_8.txt    1900_8.txt     4150_10.txt    6400_9.txt     8651_9.t
xt
10901_10.txt   1901_7.txt     4151_10.txt    6401_8.txt     8652_8.t
xt
10902_10.txt   1902_10.txt    4152_10.txt    6402_8.txt     8653_10.
txt
10903_10.txt   1903_10.txt    4153_10.txt    6403_8.txt     8654_9.t
xt
10904_10.txt   1904_7.txt     4154_10.txt    6404_8.txt     8655_9.t
xt
10905_8.txt    1905_10.txt    4155_10.txt    6405_7.txt     8656_9.t
xt
10906_10.txt   1906_10.txt    4156_10.txt    6406_9.txt     8657_9.t
xt
10907_9.txt    1907_7.txt     4157_8.txt     6407_10.txt    8658_8.t
xt
10908_10.txt   1908_9.txt     4158_10.txt    6408_10.txt    8659_7.t
xt
10909_10.txt   1909_10.txt    4159_9.txt     6409_7.txt     865_9.tx
t
1090_8.txt     190_10.txt     415_7.txt      640_10.txt     8660_7.t
xt
10910_10.txt   1910_9.txt     4160_9.txt     6410_10.txt    8661_8.t
xt
10911_7.txt    1911_10.txt    4161_8.txt     6411_10.txt    8662_8.t
xt
10912_7.txt    1912_9.txt     4162_10.txt    6412_9.txt     8663_7.t
xt
10913_7.txt    1913_10.txt    4163_9.txt     6413_7.txt     8664_8.t
xt
```

```
10914_8.txt    1914_7.txt    4164_10.txt    6414_8.txt    8665_8.t
xt
10915_9.txt    1915_10.txt   4165_8.txt     6415_8.txt    8666_7.t
xt
10916_7.txt    1916_8.txt    4166_9.txt     6416_7.txt    8667_7.t
xt
10917_8.txt    1917_9.txt    4167_10.txt    6417_7.txt    8668_9.t
xt
10918_8.txt    1918_9.txt    4168_7.txt     6418_10.txt   8669_7.t
xt
10919_10.txt   1919_8.txt    4169_7.txt     6419_10.txt   866_8.tx
t
1091_10.txt    191_9.txt     416_8.txt      641_8.txt     8670_8.t
xt
10920_10.txt   1920_10.txt   4170_8.txt     6420_7.txt    8671_7.t
xt
10921_10.txt   1921_9.txt    4171_7.txt     6421_8.txt    8672_8.t
xt
10922_10.txt   1922_9.txt    4172_10.txt    6422_7.txt    8673_7.t
xt
10923_9.txt    1923_10.txt   4173_10.txt    6423_7.txt    8674_9.t
xt
10924_10.txt   1924_10.txt   4174_9.txt     6424_10.txt   8675_8.t
xt
10925_9.txt    1925_9.txt    4175_8.txt     6425_9.txt    8676_8.t
xt
10926_9.txt    1926_10.txt   4176_10.txt    6426_8.txt    8677_9.t
xt
10927_10.txt   1927_8.txt    4177_9.txt     6427_10.txt   8678_9.t
xt
10928_7.txt    1928_10.txt   4178_10.txt    6428_8.txt    8679_8.t
xt
10929_10.txt   1929_10.txt   4179_10.txt    6429_7.txt    867_8.tx
t
1092_10.txt    192_9.txt     417_7.txt      642_10.txt    8680_9.t
xt
10930_8.txt    1930_10.txt   4180_10.txt    6430_10.txt   8681_8.t
xt
10931_7.txt    1931_8.txt    4181_9.txt     6431_8.txt    8682_7.t
xt
```

```
10932_7.txt    1932_10.txt    4182_10.txt    6432_9.txt     8683_9.t
xt
10933_10.txt   1933_8.txt     4183_10.txt    6433_7.txt     8684_10.
txt
10934_10.txt   1934_9.txt     4184_8.txt     6434_7.txt     8685_9.t
xt
10935_7.txt    1935_10.txt    4185_8.txt     6435_7.txt     8686_8.t
xt
10936_8.txt    1936_10.txt    4186_7.txt     6436_10.txt    8687_9.t
xt
10937_9.txt    1937_10.txt    4187_7.txt     6437_9.txt     8688_8.t
xt
10938_10.txt   1938_9.txt     4188_8.txt     6438_10.txt    8689_7.t
xt
10939_8.txt    1939_8.txt     4189_8.txt     6439_10.txt    868_8.tx
t
1093_8.txt     193_7.txt      418_9.txt      643_10.txt     8690_8.t
xt
10940_10.txt   1940_10.txt    4190_10.txt    6440_8.txt     8691_7.t
xt
10941_7.txt    1941_9.txt     4191_10.txt    6441_9.txt     8692_10.
txt
10942_9.txt    1942_10.txt    4192_10.txt    6442_9.txt     8693_10.
txt
10943_10.txt   1943_10.txt    4193_8.txt     6443_9.txt     8694_10.
txt
10944_8.txt    1944_8.txt     4194_10.txt    6444_8.txt     8695_9.t
xt
10945_9.txt    1945_8.txt     4195_9.txt     6445_10.txt    8696_10.
txt
10946_7.txt    1946_9.txt     4196_9.txt     6446_10.txt    8697_7.t
xt
10947_8.txt    1947_8.txt     4197_8.txt     6447_8.txt     8698_10.
txt
10948_10.txt   1948_10.txt    4198_7.txt     6448_10.txt    8699_10.
txt
10949_8.txt    1949_8.txt     4199_7.txt     6449_10.txt    869_7.tx
t
1094_9.txt     194_8.txt      419_7.txt      644_9.txt      86_10.tx
t
```

```
10950_9.txt   1950_8.txt    41_9.txt      6450_10.txt   8700_7.t
xt
10951_8.txt   1951_9.txt    4200_9.txt    6451_8.txt    8701_8.t
xt
10952_10.txt  1952_8.txt    4201_9.txt    6452_9.txt    8702_10.
txt
10953_10.txt  1953_10.txt   4202_10.txt   6453_10.txt   8703_10.
txt
10954_10.txt  1954_10.txt   4203_8.txt    6454_9.txt    8704_9.t
xt
10955_7.txt   1955_9.txt    4204_10.txt   6455_7.txt    8705_8.t
xt
10956_9.txt   1956_8.txt    4205_8.txt    6456_10.txt   8706_8.t
xt
10957_10.txt  1957_7.txt    4206_9.txt    6457_7.txt    8707_10.
txt
10958_10.txt  1958_10.txt   4207_7.txt    6458_8.txt    8708_8.t
xt
10959_10.txt  1959_7.txt    4208_10.txt   6459_10.txt   8709_8.t
xt
1095_9.txt    195_8.txt     4209_7.txt    645_10.txt    870_10.t
xt
10960_8.txt   1960_7.txt    420_7.txt     6460_10.txt   8710_7.t
xt
10961_9.txt   1961_9.txt    4210_9.txt    6461_10.txt   8711_7.t
xt
10962_10.txt  1962_10.txt   4211_8.txt    6462_10.txt   8712_8.t
xt
10963_7.txt   1963_8.txt    4212_10.txt   6463_8.txt    8713_10.
txt
10964_7.txt   1964_7.txt    4213_7.txt    6464_9.txt    8714_10.
txt
10965_9.txt   1965_7.txt    4214_10.txt   6465_9.txt    8715_10.
txt
10966_9.txt   1966_10.txt   4215_9.txt    6466_10.txt   8716_10.
txt
10967_10.txt  1967_8.txt    4216_10.txt   6467_7.txt    8717_9.t
xt
10968_7.txt   1968_8.txt    4217_9.txt    6468_7.txt    8718_9.t
xt
```

```
10969_10.txt  1969_10.txt  4218_8.txt   6469_9.txt   8719_10.txt
1096_9.txt    196_9.txt    4219_7.txt   646_9.txt    871_9.txt
10970_7.txt   1970_9.txt   421_9.txt    6470_10.txt  8720_9.txt
10971_9.txt   1971_9.txt   4220_7.txt   6471_10.txt  8721_10.txt
10972_10.txt  1972_10.txt  4221_8.txt   6472_10.txt  8722_9.txt
10973_8.txt   1973_8.txt   4222_7.txt   6473_8.txt   8723_8.txt
10974_10.txt  1974_8.txt   4223_8.txt   6474_8.txt   8724_10.txt
10975_10.txt  1975_7.txt   4224_10.txt  6475_10.txt  8725_10.txt
10976_10.txt  1976_10.txt  4225_10.txt  6476_10.txt  8726_9.txt
10977_8.txt   1977_10.txt  4226_10.txt  6477_7.txt   8727_7.txt
10978_10.txt  1978_9.txt   4227_10.txt  6478_10.txt  8728_7.txt
10979_9.txt   1979_9.txt   4228_10.txt  6479_10.txt  8729_10.txt
1097_9.txt    197_9.txt    4229_10.txt  647_10.txt   872_9.txt
10980_7.txt   1980_10.txt  422_7.txt    6480_10.txt  8730_8.txt
10981_8.txt   1981_9.txt   4230_10.txt  6481_10.txt  8731_7.txt
10982_10.txt  1982_10.txt  4231_7.txt   6482_10.txt  8732_9.txt
10983_10.txt  1983_10.txt  4232_7.txt   6483_10.txt  8733_10.txt
10984_10.txt  1984_10.txt  4233_7.txt   6484_10.txt  8734_9.txt
10985_9.txt   1985_10.txt  4234_7.txt   6485_10.txt  8735_8.txt
10986_10.txt  1986_10.txt  4235_7.txt   6486_10.txt  8736_10.txt
```

```
10987_8.txt    1987_10.txt    4236_8.txt     6487_10.txt    8737_9.t
xt
10988_10.txt   1988_9.txt     4237_10.txt    6488_10.txt    8738_8.t
xt
10989_9.txt    1989_8.txt     4238_9.txt     6489_10.txt    8739_9.t
xt
1098_9.txt     198_8.txt      4239_10.txt    648_7.txt      873_8.tx
t
10990_7.txt    1990_10.txt    423_10.txt     6490_10.txt    8740_8.t
xt
10991_9.txt    1991_10.txt    4240_9.txt     6491_10.txt    8741_10.
txt
10992_10.txt   1992_10.txt    4241_10.txt    6492_10.txt    8742_10.
txt
10993_10.txt   1993_10.txt    4242_9.txt     6493_10.txt    8743_7.t
xt
10994_8.txt    1994_10.txt    4243_8.txt     6494_10.txt    8744_7.t
xt
10995_10.txt   1995_9.txt     4244_10.txt    6495_10.txt    8745_7.t
xt
10996_8.txt    1996_10.txt    4245_7.txt     6496_10.txt    8746_10.
txt
10997_10.txt   1997_9.txt     4246_8.txt     6497_10.txt    8747_7.t
xt
10998_7.txt    1998_9.txt     4247_10.txt    6498_10.txt    8748_8.t
xt
10999_7.txt    1999_9.txt     4248_10.txt    6499_10.txt    8749_7.t
xt
1099_10.txt    199_10.txt     4249_10.txt    649_10.txt     874_9.tx
t
109_10.txt     19_10.txt      424_8.txt      64_7.txt       8750_7.t
xt
10_9.txt       1_7.txt        4250_8.txt     6500_10.txt    8751_7.t
xt
11000_10.txt   2000_10.txt    4251_9.txt     6501_10.txt    8752_9.t
xt
11001_10.txt   2001_9.txt     4252_9.txt     6502_10.txt    8753_8.t
xt
11002_8.txt    2002_7.txt     4253_10.txt    6503_10.txt    8754_8.t
xt
```

```
11003_10.txt  2003_8.txt    4254_9.txt    6504_10.txt   8755_7.t
xt
11004_8.txt   2004_10.txt   4255_9.txt    6505_10.txt   8756_9.t
xt
11005_10.txt  2005_10.txt   4256_8.txt    6506_10.txt   8757_8.t
xt
11006_7.txt   2006_7.txt    4257_10.txt   6507_10.txt   8758_10.
txt
11007_10.txt  2007_7.txt    4258_9.txt    6508_7.txt    8759_8.t
xt
11008_9.txt   2008_7.txt    4259_9.txt    6509_9.txt    875_10.t
xt
11009_7.txt   2009_10.txt   425_10.txt    650_9.txt     8760_7.t
xt
1100_7.txt    200_10.txt    4260_9.txt    6510_7.txt    8761_10.
txt
11010_10.txt  2010_8.txt    4261_9.txt    6511_10.txt   8762_8.t
xt
11011_10.txt  2011_7.txt    4262_10.txt   6512_7.txt    8763_9.t
xt
11012_9.txt   2012_8.txt    4263_8.txt    6513_9.txt    8764_8.t
xt
11013_7.txt   2013_8.txt    4264_10.txt   6514_10.txt   8765_10.
txt
11014_10.txt  2014_7.txt    4265_8.txt    6515_7.txt    8766_8.t
xt
11015_9.txt   2015_8.txt    4266_7.txt    6516_10.txt   8767_10.
txt
11016_7.txt   2016_7.txt    4267_8.txt    6517_10.txt   8768_8.t
xt
11017_9.txt   2017_10.txt   4268_10.txt   6518_10.txt   8769_7.t
xt
11018_10.txt  2018_9.txt    4269_10.txt   6519_8.txt    876_7.tx
t
11019_10.txt  2019_10.txt   426_7.txt     651_10.txt    8770_7.t
xt
1101_8.txt    201_10.txt    4270_10.txt   6520_7.txt    8771_7.t
xt
11020_8.txt   2020_7.txt    4271_10.txt   6521_7.txt    8772_8.t
xt
```

```
11021_8.txt    2021_8.txt     4272_10.txt    6522_10.txt    8773_8.t
xt
11022_9.txt    2022_9.txt     4273_10.txt    6523_10.txt    8774_8.t
xt
11023_9.txt    2023_7.txt     4274_10.txt    6524_8.txt     8775_9.t
xt
11024_9.txt    2024_9.txt     4275_10.txt    6525_7.txt     8776_10.
txt
11025_7.txt    2025_10.txt    4276_10.txt    6526_7.txt     8777_10.
txt
11026_7.txt    2026_8.txt     4277_10.txt    6527_8.txt     8778_8.t
xt
11027_10.txt   2027_10.txt    4278_10.txt    6528_9.txt     8779_8.t
xt
11028_8.txt    2028_10.txt    4279_10.txt    6529_7.txt     877_8.tx
t
11029_7.txt    2029_8.txt     427_10.txt     652_10.txt     8780_10.
txt
1102_8.txt     202_10.txt     4280_10.txt    6530_8.txt     8781_9.t
xt
11030_8.txt    2030_9.txt     4281_10.txt    6531_8.txt     8782_9.t
xt
11031_8.txt    2031_8.txt     4282_10.txt    6532_8.txt     8783_9.t
xt
11032_7.txt    2032_10.txt    4283_10.txt    6533_7.txt     8784_9.t
xt
11033_9.txt    2033_8.txt     4284_10.txt    6534_10.txt    8785_10.
txt
11034_9.txt    2034_9.txt     4285_10.txt    6535_10.txt    8786_8.t
xt
11035_8.txt    2035_7.txt     4286_10.txt    6536_7.txt     8787_8.t
xt
11036_8.txt    2036_7.txt     4287_10.txt    6537_7.txt     8788_10.
txt
11037_8.txt    2037_8.txt     4288_9.txt     6538_8.txt     8789_10.
txt
11038_10.txt   2038_7.txt     4289_7.txt     6539_9.txt     878_7.tx
t
11039_8.txt    2039_9.txt     428_7.txt      653_10.txt     8790_8.t
xt
```

```
1103_10.txt   203_7.txt     4290_9.txt    6540_7.txt    8791_10.
txt
11040_7.txt   2040_7.txt    4291_10.txt   6541_8.txt    8792_8.t
xt
11041_7.txt   2041_7.txt    4292_7.txt    6542_8.txt    8793_10.
txt
11042_7.txt   2042_7.txt    4293_8.txt    6543_9.txt    8794_9.t
xt
11043_7.txt   2043_7.txt    4294_8.txt    6544_10.txt   8795_10.
txt
11044_7.txt   2044_8.txt    4295_9.txt    6545_10.txt   8796_8.t
xt
11045_9.txt   2045_9.txt    4296_9.txt    6546_10.txt   8797_8.t
xt
11046_8.txt   2046_8.txt    4297_9.txt    6547_9.txt    8798_8.t
xt
11047_9.txt   2047_10.txt   4298_10.txt   6548_10.txt   8799_8.t
xt
11048_7.txt   2048_7.txt    4299_8.txt    6549_8.txt    879_8.tx
t
11049_8.txt   2049_7.txt    429_10.txt    654_10.txt    87_10.tx
t
1104_8.txt    204_10.txt    42_10.txt     6550_10.txt   8800_10.
txt
11050_8.txt   2050_7.txt    4300_7.txt    6551_9.txt    8801_9.t
xt
11051_8.txt   2051_8.txt    4301_10.txt   6552_9.txt    8802_7.t
xt
11052_9.txt   2052_10.txt   4302_8.txt    6553_10.txt   8803_10.
txt
11053_10.txt  2053_10.txt   4303_10.txt   6554_7.txt    8804_10.
txt
11054_7.txt   2054_9.txt    4304_9.txt    6555_8.txt    8805_9.t
xt
11055_10.txt  2055_10.txt   4305_10.txt   6556_9.txt    8806_9.t
xt
11056_10.txt  2056_9.txt    4306_10.txt   6557_10.txt   8807_9.t
xt
11057_10.txt  2057_10.txt   4307_10.txt   6558_8.txt    8808_9.t
xt
```

```
11058_10.txt  2058_9.txt   4308_8.txt   6559_10.txt  8809_10.
txt
11059_9.txt   2059_7.txt   4309_8.txt   655_10.txt   880_10.t
xt
1105_8.txt    205_8.txt    430_7.txt    6560_7.txt   8810_9.t
xt
11060_10.txt  2060_8.txt   4310_9.txt   6561_10.txt  8811_7.t
xt
11061_8.txt   2061_9.txt   4311_9.txt   6562_10.txt  8812_10.
txt
11062_10.txt  2062_10.txt  4312_10.txt  6563_7.txt   8813_9.t
xt
11063_10.txt  2063_8.txt   4313_9.txt   6564_7.txt   8814_10.
txt
11064_7.txt   2064_8.txt   4314_10.txt  6565_8.txt   8815_7.t
xt
11065_8.txt   2065_7.txt   4315_9.txt   6566_8.txt   8816_7.t
xt
11066_8.txt   2066_7.txt   4316_10.txt  6567_10.txt  8817_8.t
xt
11067_7.txt   2067_9.txt   4317_10.txt  6568_9.txt   8818_10.
txt
11068_9.txt   2068_8.txt   4318_10.txt  6569_7.txt   8819_10.
txt
11069_10.txt  2069_9.txt   4319_9.txt   656_10.txt   881_8.tx
t
1106_8.txt    206_10.txt   431_8.txt    6570_8.txt   8820_7.t
xt
11070_9.txt   2070_9.txt   4320_8.txt   6571_10.txt  8821_9.t
xt
11071_10.txt  2071_9.txt   4321_8.txt   6572_8.txt   8822_10.
txt
11072_8.txt   2072_10.txt  4322_10.txt  6573_7.txt   8823_8.t
xt
11073_8.txt   2073_7.txt   4323_9.txt   6574_8.txt   8824_8.t
xt
11074_7.txt   2074_10.txt  4324_7.txt   6575_8.txt   8825_9.t
xt
11075_10.txt  2075_8.txt   4325_10.txt  6576_8.txt   8826_9.t
xt
```

```
11076_8.txt    2076_9.txt     4326_7.txt     6577_8.txt     8827_8.t
xt
11077_8.txt    2077_10.txt    4327_8.txt     6578_7.txt     8828_10.
txt
11078_10.txt   2078_9.txt     4328_10.txt    6579_10.txt    8829_9.t
xt
11079_8.txt    2079_10.txt    4329_7.txt     657_10.txt     882_8.tx
t
1107_10.txt    207_8.txt      432_8.txt      6580_8.txt     8830_8.t
xt
11080_10.txt   2080_9.txt     4330_10.txt    6581_7.txt     8831_8.t
xt
11081_10.txt   2081_7.txt     4331_7.txt     6582_7.txt     8832_7.t
xt
11082_10.txt   2082_10.txt    4332_10.txt    6583_7.txt     8833_9.t
xt
11083_10.txt   2083_7.txt     4333_10.txt    6584_8.txt     8834_9.t
xt
11084_10.txt   2084_8.txt     4334_10.txt    6585_10.txt    8835_9.t
xt
11085_10.txt   2085_7.txt     4335_9.txt     6586_10.txt    8836_10.
txt
11086_7.txt    2086_10.txt    4336_10.txt    6587_9.txt     8837_8.t
xt
11087_8.txt    2087_10.txt    4337_7.txt     6588_9.txt     8838_9.t
xt
11088_7.txt    2088_8.txt     4338_10.txt    6589_10.txt    8839_8.t
xt
11089_10.txt   2089_10.txt    4339_10.txt    658_10.txt     883_9.tx
t
1108_7.txt     208_9.txt      433_10.txt     6590_9.txt     8840_10.
txt
11090_8.txt    2090_10.txt    4340_8.txt     6591_8.txt     8841_8.t
xt
11091_8.txt    2091_9.txt     4341_8.txt     6592_10.txt    8842_9.t
xt
11092_8.txt    2092_10.txt    4342_10.txt    6593_7.txt     8843_7.t
xt
11093_10.txt   2093_7.txt     4343_9.txt     6594_10.txt    8844_10.
txt
```

```
11094_9.txt   2094_10.txt   4344_9.txt    6595_9.txt    8845_8.t
xt
11095_7.txt   2095_9.txt    4345_10.txt   6596_8.txt    8846_9.t
xt
11096_10.txt  2096_10.txt   4346_10.txt   6597_9.txt    8847_9.t
xt
11097_9.txt   2097_9.txt    4347_8.txt    6598_8.txt    8848_7.t
xt
11098_10.txt  2098_10.txt   4348_7.txt    6599_8.txt    8849_9.t
xt
11099_10.txt  2099_10.txt   4349_10.txt   659_10.txt    884_8.tx
t
1109_7.txt    209_8.txt     434_8.txt     65_10.txt     8850_8.t
xt
110_10.txt    20_9.txt      4350_10.txt   6600_9.txt    8851_7.t
xt
11100_10.txt  2100_7.txt    4351_10.txt   6601_9.txt    8852_9.t
xt
11101_10.txt  2101_7.txt    4352_10.txt   6602_7.txt    8853_7.t
xt
11102_10.txt  2102_10.txt   4353_9.txt    6603_8.txt    8854_10.
txt
11103_10.txt  2103_7.txt    4354_8.txt    6604_7.txt    8855_10.
txt
11104_10.txt  2104_7.txt    4355_10.txt   6605_7.txt    8856_8.t
xt
11105_10.txt  2105_8.txt    4356_8.txt    6606_9.txt    8857_10.
txt
11106_10.txt  2106_10.txt   4357_10.txt   6607_9.txt    8858_10.
txt
11107_10.txt  2107_7.txt    4358_10.txt   6608_7.txt    8859_10.
txt
11108_10.txt  2108_10.txt   4359_10.txt   6609_8.txt    885_8.tx
t
11109_9.txt   2109_9.txt    435_8.txt     660_9.txt     8860_8.t
xt
1110_9.txt    210_10.txt    4360_10.txt   6610_8.txt    8861_9.t
xt
11110_9.txt   2110_9.txt    4361_10.txt   6611_7.txt    8862_8.t
xt
```

```
11111_9.txt    2111_7.txt     4362_7.txt     6612_8.txt     8863_8.t
xt
11112_7.txt    2112_9.txt     4363_8.txt     6613_7.txt     8864_10.
txt
11113_9.txt    2113_10.txt    4364_7.txt     6614_9.txt     8865_10.
txt
11114_10.txt   2114_10.txt    4365_9.txt     6615_8.txt     8866_9.t
xt
11115_10.txt   2115_10.txt    4366_9.txt     6616_7.txt     8867_10.
txt
11116_10.txt   2116_10.txt    4367_9.txt     6617_8.txt     8868_10.
txt
11117_10.txt   2117_10.txt    4368_10.txt    6618_8.txt     8869_7.t
xt
11118_10.txt   2118_9.txt     4369_9.txt     6619_9.txt     886_8.tx
t
11119_10.txt   2119_10.txt    436_10.txt     661_9.txt      8870_8.t
xt
1111_10.txt    211_9.txt      4370_10.txt    6620_10.txt    8871_10.
txt
11120_10.txt   2120_8.txt     4371_8.txt     6621_8.txt     8872_8.t
xt
11121_10.txt   2121_10.txt    4372_10.txt    6622_10.txt    8873_8.t
xt
11122_10.txt   2122_7.txt     4373_10.txt    6623_7.txt     8874_8.t
xt
11123_10.txt   2123_10.txt    4374_10.txt    6624_9.txt     8875_9.t
xt
11124_10.txt   2124_10.txt    4375_9.txt     6625_7.txt     8876_10.
txt
11125_10.txt   2125_10.txt    4376_9.txt     6626_7.txt     8877_9.t
xt
11126_10.txt   2126_10.txt    4377_10.txt    6627_7.txt     8878_9.t
xt
11127_9.txt    2127_9.txt     4378_9.txt     6628_7.txt     8879_7.t
xt
11128_10.txt   2128_8.txt     4379_8.txt     6629_10.txt    887_10.t
xt
11129_10.txt   2129_8.txt     437_9.txt      662_8.txt      8880_8.t
xt
```

```
1112_8.txt    212_9.txt     4380_8.txt    6630_10.txt   8881_8.t
xt
11130_9.txt   2130_10.txt   4381_10.txt   6631_7.txt    8882_10.
txt
11131_9.txt   2131_9.txt    4382_8.txt    6632_7.txt    8883_8.t
xt
11132_10.txt  2132_9.txt    4383_9.txt    6633_10.txt   8884_9.t
xt
11133_10.txt  2133_10.txt   4384_10.txt   6634_10.txt   8885_10.
txt
11134_9.txt   2134_10.txt   4385_9.txt    6635_10.txt   8886_10.
txt
11135_9.txt   2135_8.txt    4386_9.txt    6636_8.txt    8887_9.t
xt
11136_10.txt  2136_9.txt    4387_9.txt    6637_8.txt    8888_7.t
xt
11137_7.txt   2137_8.txt    4388_9.txt    6638_10.txt   8889_8.t
xt
11138_10.txt  2138_9.txt    4389_10.txt   6639_10.txt   888_8.tx
t
11139_8.txt   2139_9.txt    438_9.txt     663_8.txt     8890_10.
txt
1113_10.txt   213_9.txt     4390_8.txt    6640_8.txt    8891_10.
txt
11140_9.txt   2140_10.txt   4391_8.txt    6641_8.txt    8892_10.
txt
11141_10.txt  2141_10.txt   4392_9.txt    6642_10.txt   8893_9.t
xt
11142_8.txt   2142_8.txt    4393_7.txt    6643_7.txt    8894_10.
txt
11143_10.txt  2143_9.txt    4394_9.txt    6644_8.txt    8895_10.
txt
11144_8.txt   2144_8.txt    4395_10.txt   6645_8.txt    8896_10.
txt
11145_8.txt   2145_7.txt    4396_10.txt   6646_10.txt   8897_10.
txt
11146_8.txt   2146_9.txt    4397_7.txt    6647_8.txt    8898_8.t
xt
11147_9.txt   2147_10.txt   4398_9.txt    6648_7.txt    8899_8.t
xt
```

```
11148_9.txt    2148_10.txt    4399_9.txt    6649_8.txt    889_10.t
xt
11149_10.txt   2149_10.txt    439_9.txt     664_7.txt     88_9.txt
1114_8.txt     214_7.txt      43_10.txt     6650_8.txt    8900_10.
txt
11150_10.txt   2150_10.txt    4400_10.txt   6651_10.txt   8901_8.t
xt
11151_8.txt    2151_10.txt    4401_9.txt    6652_10.txt   8902_7.t
xt
11152_10.txt   2152_8.txt     4402_8.txt    6653_8.txt    8903_7.t
xt
11153_10.txt   2153_9.txt     4403_10.txt   6654_7.txt    8904_7.t
xt
11154_10.txt   2154_10.txt    4404_9.txt    6655_7.txt    8905_10.
txt
11155_10.txt   2155_10.txt    4405_8.txt    6656_10.txt   8906_8.t
xt
11156_8.txt    2156_10.txt    4406_9.txt    6657_9.txt    8907_8.t
xt
11157_7.txt    2157_10.txt    4407_8.txt    6658_10.txt   8908_10.
txt
11158_8.txt    2158_10.txt    4408_8.txt    6659_10.txt   8909_9.t
xt
11159_8.txt    2159_10.txt    4409_10.txt   665_9.txt     890_9.tx
t
1115_9.txt     215_8.txt      440_10.txt    6660_8.txt    8910_10.
txt
11160_9.txt    2160_8.txt     4410_10.txt   6661_7.txt    8911_8.t
xt
11161_8.txt    2161_10.txt    4411_9.txt    6662_7.txt    8912_10.
txt
11162_10.txt   2162_10.txt    4412_9.txt    6663_10.txt   8913_9.t
xt
11163_9.txt    2163_10.txt    4413_8.txt    6664_9.txt    8914_8.t
xt
11164_8.txt    2164_10.txt    4414_9.txt    6665_10.txt   8915_10.
txt
11165_8.txt    2165_7.txt     4415_10.txt   6666_7.txt    8916_8.t
xt
11166_7.txt    2166_7.txt     4416_9.txt    6667_8.txt    8917_7.t
```

```
xt
11167_9.txt   2167_10.txt   4417_9.txt    6668_8.txt    8918_10.
txt
11168_8.txt   2168_8.txt    4418_10.txt   6669_10.txt   8919_7.t
xt
11169_7.txt   2169_7.txt    4419_7.txt    666_10.txt    891_10.t
xt
1116_9.txt    216_8.txt     441_9.txt     6670_8.txt    8920_10.
txt
11170_10.txt  2170_9.txt    4420_8.txt    6671_10.txt   8921_8.t
xt
11171_7.txt   2171_10.txt   4421_8.txt    6672_10.txt   8922_7.t
xt
11172_10.txt  2172_7.txt    4422_8.txt    6673_7.txt    8923_8.t
xt
11173_7.txt   2173_10.txt   4423_8.txt    6674_9.txt    8924_10.
txt
11174_8.txt   2174_8.txt    4424_7.txt    6675_8.txt    8925_7.t
xt
11175_8.txt   2175_9.txt    4425_7.txt    6676_9.txt    8926_10.
txt
11176_9.txt   2176_8.txt    4426_7.txt    6677_9.txt    8927_8.t
xt
11177_8.txt   2177_8.txt    4427_10.txt   6678_7.txt    8928_10.
txt
11178_7.txt   2178_9.txt    4428_10.txt   6679_8.txt    8929_8.t
xt
11179_9.txt   2179_8.txt    4429_7.txt    667_8.txt     892_10.t
xt
1117_10.txt   217_8.txt     442_9.txt     6680_7.txt    8930_9.t
xt
11180_7.txt   2180_8.txt    4430_8.txt    6681_10.txt   8931_9.t
xt
11181_10.txt  2181_10.txt   4431_10.txt   6682_10.txt   8932_9.t
xt
11182_10.txt  2182_9.txt    4432_10.txt   6683_8.txt    8933_7.t
xt
11183_7.txt   2183_8.txt    4433_7.txt    6684_10.txt   8934_10.
txt
11184_7.txt   2184_7.txt    4434_7.txt    6685_10.txt   8935_8.t
```

```
xt
11185_9.txt   2185_9.txt    4435_7.txt    6686_9.txt    8936_8.t
xt
11186_7.txt   2186_8.txt    4436_7.txt    6687_9.txt    8937_9.t
xt
11187_9.txt   2187_10.txt   4437_7.txt    6688_9.txt    8938_9.t
xt
11188_10.txt  2188_10.txt   4438_8.txt    6689_8.txt    8939_9.t
xt
11189_8.txt   2189_10.txt   4439_8.txt    668_7.txt     893_8.tx
t
1118_10.txt   218_9.txt     443_10.txt    6690_10.txt   8940_9.t
xt
11190_10.txt  2190_10.txt   4440_7.txt    6691_9.txt    8941_10.
txt
11191_10.txt  2191_10.txt   4441_7.txt    6692_9.txt    8942_8.t
xt
11192_10.txt  2192_9.txt    4442_9.txt    6693_10.txt   8943_7.t
xt
11193_7.txt   2193_10.txt   4443_8.txt    6694_10.txt   8944_8.t
xt
11194_10.txt  2194_10.txt   4444_10.txt   6695_10.txt   8945_10.
txt
11195_7.txt   2195_10.txt   4445_7.txt    6696_7.txt    8946_8.t
xt
11196_7.txt   2196_8.txt    4446_10.txt   6697_10.txt   8947_9.t
xt
11197_8.txt   2197_7.txt    4447_7.txt    6698_10.txt   8948_7.t
xt
11198_8.txt   2198_8.txt    4448_10.txt   6699_8.txt    8949_7.t
xt
11199_9.txt   2199_7.txt    4449_8.txt    669_7.txt     894_9.tx
t
1119_10.txt   219_8.txt     444_10.txt    66_8.txt      8950_7.t
xt
111_10.txt    21_7.txt      4450_9.txt    6700_8.txt    8951_7.t
xt
11200_8.txt   2200_9.txt    4451_9.txt    6701_9.txt    8952_7.t
xt
11201_7.txt   2201_10.txt   4452_8.txt    6702_8.txt    8953_8.t
```

```
xt
11202_10.txt  2202_10.txt   4453_10.txt   6703_8.txt    8954_8.t
xt
11203_8.txt   2203_8.txt    4454_9.txt    6704_9.txt    8955_8.t
xt
11204_9.txt   2204_10.txt   4455_8.txt    6705_10.txt   8956_8.t
xt
11205_10.txt  2205_8.txt    4456_7.txt    6706_8.txt    8957_7.t
xt
11206_10.txt  2206_7.txt    4457_8.txt    6707_9.txt    8958_10.
txt
11207_10.txt  2207_9.txt    4458_7.txt    6708_10.txt   8959_8.t
xt
11208_8.txt   2208_7.txt    4459_7.txt    6709_10.txt   895_10.t
xt
11209_8.txt   2209_8.txt    445_10.txt    670_7.txt     8960_10.
txt
1120_7.txt    220_10.txt    4460_9.txt    6710_10.txt   8961_9.t
xt
11210_7.txt   2210_9.txt    4461_8.txt    6711_10.txt   8962_9.t
xt
11211_8.txt   2211_8.txt    4462_9.txt    6712_9.txt    8963_8.t
xt
11212_8.txt   2212_10.txt   4463_7.txt    6713_10.txt   8964_8.t
xt
11213_7.txt   2213_9.txt    4464_10.txt   6714_10.txt   8965_8.t
xt
11214_7.txt   2214_10.txt   4465_7.txt    6715_10.txt   8966_10.
txt
11215_10.txt  2215_8.txt    4466_8.txt    6716_8.txt    8967_8.t
xt
11216_7.txt   2216_9.txt    4467_7.txt    6717_7.txt    8968_8.t
xt
11217_7.txt   2217_7.txt    4468_7.txt    6718_10.txt   8969_7.t
xt
11218_9.txt   2218_7.txt    4469_9.txt    6719_8.txt    896_10.t
xt
11219_7.txt   2219_10.txt   446_10.txt    671_7.txt     8970_10.
txt
1121_7.txt    221_9.txt     4470_8.txt    6720_9.txt    8971_9.t
```

```
xt
11220_10.txt  2220_9.txt    4471_8.txt    6721_10.txt   8972_10.
txt
11221_10.txt  2221_8.txt    4472_10.txt   6722_9.txt    8973_9.t
xt
11222_8.txt   2222_10.txt   4473_8.txt    6723_10.txt   8974_10.
txt
11223_9.txt   2223_8.txt    4474_10.txt   6724_8.txt    8975_10.
txt
11224_8.txt   2224_10.txt   4475_8.txt    6725_9.txt    8976_7.t
xt
11225_10.txt  2225_10.txt   4476_9.txt    6726_9.txt    8977_7.t
xt
11226_7.txt   2226_10.txt   4477_7.txt    6727_9.txt    8978_10.
txt
11227_9.txt   2227_7.txt    4478_8.txt    6728_8.txt    8979_10.
txt
11228_10.txt  2228_7.txt    4479_8.txt    6729_10.txt   897_10.t
xt
11229_10.txt  2229_7.txt    447_10.txt    672_9.txt     8980_10.
txt
1122_7.txt    222_10.txt    4480_9.txt    6730_10.txt   8981_7.t
xt
11230_10.txt  2230_10.txt   4481_8.txt    6731_10.txt   8982_8.t
xt
11231_10.txt  2231_10.txt   4482_7.txt    6732_10.txt   8983_8.t
xt
11232_10.txt  2232_7.txt    4483_8.txt    6733_10.txt   8984_10.
txt
11233_10.txt  2233_10.txt   4484_8.txt    6734_10.txt   8985_9.t
xt
11234_10.txt  2234_9.txt    4485_10.txt   6735_9.txt    8986_8.t
xt
11235_8.txt   2235_10.txt   4486_7.txt    6736_10.txt   8987_9.t
xt
11236_8.txt   2236_9.txt    4487_9.txt    6737_7.txt    8988_10.
txt
11237_8.txt   2237_7.txt    4488_10.txt   6738_7.txt    8989_9.t
xt
11238_7.txt   2238_9.txt    4489_9.txt    6739_7.txt    898_10.t
```

```
xt
11239_8.txt   2239_7.txt    448_10.txt    673_8.txt     8990_10.
txt
1123_10.txt   223_9.txt     4490_7.txt    6740_8.txt    8991_10.
txt
11240_7.txt   2240_7.txt    4491_10.txt   6741_7.txt    8992_9.t
xt
11241_10.txt  2241_7.txt    4492_10.txt   6742_8.txt    8993_7.t
xt
11242_9.txt   2242_7.txt    4493_9.txt    6743_9.txt    8994_8.t
xt
11243_10.txt  2243_7.txt    4494_8.txt    6744_8.txt    8995_9.t
xt
11244_10.txt  2244_9.txt    4495_9.txt    6745_8.txt    8996_8.t
xt
11245_10.txt  2245_7.txt    4496_8.txt    6746_8.txt    8997_10.
txt
11246_8.txt   2246_10.txt   4497_7.txt    6747_8.txt    8998_10.
txt
11247_8.txt   2247_10.txt   4498_8.txt    6748_8.txt    8999_8.t
xt
11248_10.txt  2248_7.txt    4499_9.txt    6749_9.txt    899_7.tx
t
11249_7.txt   2249_7.txt    449_10.txt    674_10.txt    89_7.txt
1124_10.txt   224_10.txt    44_8.txt      6750_8.txt    8_7.txt
11250_9.txt   2250_8.txt    4500_10.txt   6751_8.txt    9000_10.
txt
11251_10.txt  2251_8.txt    4501_7.txt    6752_7.txt    9001_8.t
xt
11252_9.txt   2252_10.txt   4502_7.txt    6753_7.txt    9002_9.t
xt
11253_9.txt   2253_8.txt    4503_7.txt    6754_8.txt    9003_10.
txt
11254_8.txt   2254_8.txt    4504_9.txt    6755_7.txt    9004_7.t
xt
11255_10.txt  2255_8.txt    4505_8.txt    6756_10.txt   9005_8.t
xt
11256_10.txt  2256_8.txt    4506_10.txt   6757_8.txt    9006_7.t
xt
11257_7.txt   2257_7.txt    4507_7.txt    6758_10.txt   9007_10.
```

```
txt
11258_8.txt   2258_7.txt    4508_7.txt    6759_7.txt    9008_10.
txt
11259_10.txt  2259_9.txt    4509_10.txt   675_9.txt     9009_10.
txt
1125_8.txt    225_9.txt     450_10.txt    6760_9.txt    900_10.t
xt
11260_7.txt   2260_10.txt   4510_7.txt    6761_9.txt    9010_7.t
xt
11261_9.txt   2261_10.txt   4511_9.txt    6762_9.txt    9011_9.t
xt
11262_7.txt   2262_10.txt   4512_9.txt    6763_8.txt    9012_7.t
xt
11263_10.txt  2263_10.txt   4513_8.txt    6764_9.txt    9013_7.t
xt
11264_10.txt  2264_9.txt    4514_10.txt   6765_9.txt    9014_10.
txt
11265_10.txt  2265_7.txt    4515_10.txt   6766_8.txt    9015_8.t
xt
11266_10.txt  2266_8.txt    4516_10.txt   6767_8.txt    9016_10.
txt
11267_10.txt  2267_8.txt    4517_9.txt    6768_8.txt    9017_8.t
xt
11268_8.txt   2268_10.txt   4518_9.txt    6769_9.txt    9018_8.t
xt
11269_8.txt   2269_10.txt   4519_9.txt    676_8.txt     9019_8.t
xt
1126_9.txt    226_10.txt    451_10.txt    6770_10.txt   901_8.tx
t
11270_10.txt  2270_9.txt    4520_7.txt    6771_7.txt    9020_7.t
xt
11271_7.txt   2271_8.txt    4521_10.txt   6772_10.txt   9021_10.
txt
11272_10.txt  2272_7.txt    4522_9.txt    6773_10.txt   9022_10.
txt
11273_9.txt   2273_9.txt    4523_10.txt   6774_10.txt   9023_10.
txt
11274_9.txt   2274_9.txt    4524_9.txt    6775_8.txt    9024_10.
txt
11275_10.txt  2275_7.txt    4525_8.txt    6776_7.txt    9025_9.t
```

```
xt
11276_10.txt  2276_7.txt    4526_10.txt   6777_8.txt    9026_9.t
xt
11277_10.txt  2277_10.txt   4527_9.txt    6778_8.txt    9027_7.t
xt
11278_8.txt   2278_10.txt   4528_8.txt    6779_8.txt    9028_10.
txt
11279_10.txt  2279_10.txt   4529_8.txt    677_8.txt     9029_10.
txt
1127_10.txt   227_10.txt    452_10.txt    6780_10.txt   902_9.tx
t
11280_8.txt   2280_7.txt    4530_7.txt    6781_10.txt   9030_10.
txt
11281_8.txt   2281_9.txt    4531_8.txt    6782_9.txt    9031_10.
txt
11282_7.txt   2282_10.txt   4532_7.txt    6783_7.txt    9032_8.t
xt
11283_8.txt   2283_9.txt    4533_8.txt    6784_8.txt    9033_9.t
xt
11284_8.txt   2284_10.txt   4534_9.txt    6785_8.txt    9034_10.
txt
11285_7.txt   2285_10.txt   4535_7.txt    6786_10.txt   9035_10.
txt
11286_10.txt  2286_10.txt   4536_7.txt    6787_9.txt    9036_10.
txt
11287_8.txt   2287_8.txt    4537_8.txt    6788_9.txt    9037_8.t
xt
11288_8.txt   2288_9.txt    4538_7.txt    6789_8.txt    9038_8.t
xt
11289_10.txt  2289_9.txt    4539_9.txt    678_8.txt     9039_10.
txt
1128_10.txt   228_7.txt     453_10.txt    6790_10.txt   903_8.tx
t
11290_10.txt  2290_10.txt   4540_7.txt    6791_10.txt   9040_9.t
xt
11291_10.txt  2291_7.txt    4541_10.txt   6792_10.txt   9041_8.t
xt
11292_10.txt  2292_8.txt    4542_9.txt    6793_10.txt   9042_10.
txt
11293_7.txt   2293_7.txt    4543_7.txt    6794_10.txt   9043_7.t
```

```
xt
11294_10.txt  2294_8.txt    4544_10.txt   6795_9.txt    9044_9.t
xt
11295_9.txt   2295_7.txt    4545_10.txt   6796_8.txt    9045_9.t
xt
11296_8.txt   2296_9.txt    4546_10.txt   6797_8.txt    9046_8.t
xt
11297_7.txt   2297_7.txt    4547_8.txt    6798_9.txt    9047_10.
txt
11298_8.txt   2298_7.txt    4548_7.txt    6799_9.txt    9048_8.t
xt
11299_8.txt   2299_7.txt    4549_7.txt    679_10.txt    9049_8.t
xt
1129_10.txt   229_10.txt    454_8.txt     67_10.txt     904_10.t
xt
112_10.txt    22_8.txt      4550_10.txt   6800_9.txt    9050_8.t
xt
11300_8.txt   2300_10.txt   4551_10.txt   6801_9.txt    9051_10.
txt
11301_7.txt   2301_10.txt   4552_9.txt    6802_9.txt    9052_8.t
xt
11302_8.txt   2302_9.txt    4553_10.txt   6803_8.txt    9053_10.
txt
11303_9.txt   2303_8.txt    4554_8.txt    6804_7.txt    9054_9.t
xt
11304_7.txt   2304_9.txt    4555_9.txt    6805_10.txt   9055_10.
txt
11305_8.txt   2305_10.txt   4556_8.txt    6806_7.txt    9056_8.t
xt
11306_8.txt   2306_10.txt   4557_8.txt    6807_10.txt   9057_7.t
xt
11307_10.txt  2307_9.txt    4558_9.txt    6808_8.txt    9058_7.t
xt
11308_9.txt   2308_10.txt   4559_10.txt   6809_7.txt    9059_7.t
xt
11309_9.txt   2309_9.txt    455_10.txt    680_8.txt     905_10.t
xt
1130_10.txt   230_9.txt     4560_10.txt   6810_7.txt    9060_8.t
xt
11310_10.txt  2310_9.txt    4561_10.txt   6811_10.txt   9061_7.t
```

```
xt
11311_10.txt  2311_10.txt   4562_9.txt    6812_7.txt    9062_7.t
xt
11312_9.txt   2312_9.txt    4563_8.txt    6813_10.txt   9063_8.t
xt
11313_10.txt  2313_10.txt   4564_10.txt   6814_7.txt    9064_7.t
xt
11314_8.txt   2314_7.txt    4565_8.txt    6815_8.txt    9065_10.
txt
11315_10.txt  2315_10.txt   4566_9.txt    6816_7.txt    9066_8.t
xt
11316_8.txt   2316_10.txt   4567_10.txt   6817_10.txt   9067_8.t
xt
11317_9.txt   2317_10.txt   4568_10.txt   6818_10.txt   9068_9.t
xt
11318_9.txt   2318_10.txt   4569_7.txt    6819_8.txt    9069_7.t
xt
11319_8.txt   2319_8.txt    456_10.txt    681_9.txt     906_10.t
xt
1131_7.txt    231_10.txt    4570_10.txt   6820_7.txt    9070_7.t
xt
11320_10.txt  2320_10.txt   4571_9.txt    6821_10.txt   9071_8.t
xt
11321_8.txt   2321_9.txt    4572_10.txt   6822_7.txt    9072_9.t
xt
11322_9.txt   2322_10.txt   4573_10.txt   6823_10.txt   9073_10.
txt
11323_10.txt  2323_8.txt    4574_10.txt   6824_10.txt   9074_8.t
xt
11324_10.txt  2324_8.txt    4575_8.txt    6825_10.txt   9075_10.
txt
11325_10.txt  2325_8.txt    4576_10.txt   6826_9.txt    9076_8.t
xt
11326_10.txt  2326_8.txt    4577_10.txt   6827_10.txt   9077_9.t
xt
11327_10.txt  2327_10.txt   4578_9.txt    6828_10.txt   9078_7.t
xt
11328_10.txt  2328_10.txt   4579_9.txt    6829_8.txt    9079_10.
txt
11329_9.txt   2329_10.txt   457_10.txt    682_10.txt    907_8.tx
```

```
t
1132_10.txt   232_10.txt    4580_8.txt    6830_10.txt   9080_7.t
xt
11330_8.txt   2330_8.txt    4581_10.txt   6831_9.txt    9081_8.t
xt
11331_8.txt   2331_10.txt   4582_7.txt    6832_10.txt   9082_10.
txt
11332_8.txt   2332_10.txt   4583_7.txt    6833_10.txt   9083_8.t
xt
11333_9.txt   2333_10.txt   4584_10.txt   6834_10.txt   9084_7.t
xt
11334_10.txt  2334_9.txt    4585_10.txt   6835_10.txt   9085_8.t
xt
11335_9.txt   2335_10.txt   4586_7.txt    6836_9.txt    9086_7.t
xt
11336_7.txt   2336_10.txt   4587_10.txt   6837_10.txt   9087_7.t
xt
11337_10.txt  2337_10.txt   4588_9.txt    6838_7.txt    9088_8.t
xt
11338_10.txt  2338_10.txt   4589_10.txt   6839_10.txt   9089_9.t
xt
11339_10.txt  2339_8.txt    458_10.txt    683_10.txt    908_8.tx
t
1133_10.txt   233_7.txt     4590_10.txt   6840_10.txt   9090_9.t
xt
11340_7.txt   2340_10.txt   4591_9.txt    6841_7.txt    9091_10.
txt
11341_10.txt  2341_8.txt    4592_9.txt    6842_7.txt    9092_7.t
xt
11342_9.txt   2342_8.txt    4593_10.txt   6843_9.txt    9093_10.
txt
11343_8.txt   2343_10.txt   4594_10.txt   6844_10.txt   9094_10.
txt
11344_7.txt   2344_9.txt    4595_10.txt   6845_10.txt   9095_10.
txt
11345_7.txt   2345_9.txt    4596_10.txt   6846_10.txt   9096_10.
txt
11346_10.txt  2346_9.txt    4597_8.txt    6847_10.txt   9097_10.
txt
11347_10.txt  2347_10.txt   4598_9.txt    6848_7.txt    9098_10.
```

```
txt
11348_10.txt  2348_10.txt   4599_10.txt   6849_7.txt    9099_10.
txt
11349_10.txt  2349_9.txt    459_10.txt    684_10.txt    909_8.tx
t
1134_9.txt    234_10.txt    45_10.txt     6850_7.txt    90_7.txt
11350_9.txt   2350_9.txt    4600_10.txt   6851_8.txt    9100_7.t
xt
11351_9.txt   2351_10.txt   4601_7.txt    6852_8.txt    9101_9.t
xt
11352_9.txt   2352_9.txt    4602_8.txt    6853_10.txt   9102_8.t
xt
11353_9.txt   2353_10.txt   4603_10.txt   6854_10.txt   9103_10.
txt
11354_10.txt  2354_10.txt   4604_10.txt   6855_10.txt   9104_10.
txt
11355_8.txt   2355_7.txt    4605_8.txt    6856_10.txt   9105_9.t
xt
11356_9.txt   2356_10.txt   4606_7.txt    6857_10.txt   9106_9.t
xt
11357_10.txt  2357_10.txt   4607_10.txt   6858_7.txt    9107_7.t
xt
11358_9.txt   2358_9.txt    4608_9.txt    6859_7.txt    9108_10.
txt
11359_9.txt   2359_10.txt   4609_8.txt    685_8.txt     9109_7.t
xt
1135_9.txt    235_10.txt    460_9.txt     6860_8.txt    910_10.t
xt
11360_7.txt   2360_10.txt   4610_10.txt   6861_8.txt    9110_8.t
xt
11361_10.txt  2361_10.txt   4611_10.txt   6862_10.txt   9111_10.
txt
11362_9.txt   2362_9.txt    4612_10.txt   6863_7.txt    9112_9.t
xt
11363_8.txt   2363_7.txt    4613_10.txt   6864_7.txt    9113_8.t
xt
11364_10.txt  2364_10.txt   4614_7.txt    6865_7.txt    9114_10.
txt
11365_9.txt   2365_10.txt   4615_10.txt   6866_7.txt    9115_10.
txt
```

```
11366_8.txt   2366_7.txt    4616_10.txt   6867_9.txt    9116_9.txt
11367_10.txt  2367_8.txt    4617_10.txt   6868_10.txt   9117_10.txt
11368_10.txt  2368_8.txt    4618_10.txt   6869_8.txt    9118_10.txt
11369_8.txt   2369_10.txt   4619_7.txt    686_9.txt     9119_10.txt
1136_8.txt    236_9.txt     461_8.txt     6870_8.txt    911_10.txt
11370_9.txt   2370_10.txt   4620_10.txt   6871_8.txt    9120_10.txt
11371_10.txt  2371_8.txt    4621_9.txt    6872_7.txt    9121_10.txt
11372_7.txt   2372_8.txt    4622_8.txt    6873_10.txt   9122_7.txt
11373_8.txt   2373_7.txt    4623_7.txt    6874_8.txt    9123_10.txt
11374_9.txt   2374_9.txt    4624_8.txt    6875_10.txt   9124_10.txt
11375_9.txt   2375_8.txt    4625_10.txt   6876_8.txt    9125_10.txt
11376_10.txt  2376_7.txt    4626_8.txt    6877_7.txt    9126_9.txt
11377_9.txt   2377_10.txt   4627_7.txt    6878_8.txt    9127_7.txt
11378_8.txt   2378_8.txt    4628_9.txt    6879_10.txt   9128_9.txt
11379_7.txt   2379_8.txt    4629_10.txt   687_9.txt     9129_8.txt
1137_8.txt    237_10.txt    462_8.txt     6880_7.txt    912_8.txt
11380_7.txt   2380_9.txt    4630_9.txt    6881_9.txt    9130_9.txt
11381_10.txt  2381_9.txt    4631_10.txt   6882_9.txt    9131_9.txt
11382_8.txt   2382_7.txt    4632_10.txt   6883_9.txt    9132_10.txt
11383_7.txt   2383_9.txt    4633_9.txt    6884_10.txt   9133_9.txt
```

```
11384_7.txt    2384_10.txt    4634_10.txt    6885_9.txt     9134_7.t
xt
11385_8.txt    2385_9.txt     4635_7.txt     6886_8.txt     9135_9.t
xt
11386_8.txt    2386_8.txt     4636_8.txt     6887_8.txt     9136_8.t
xt
11387_8.txt    2387_10.txt    4637_8.txt     6888_9.txt     9137_10.
txt
11388_8.txt    2388_10.txt    4638_10.txt    6889_10.txt    9138_10.
txt
11389_7.txt    2389_10.txt    4639_10.txt    688_9.txt      9139_8.t
xt
1138_10.txt    238_10.txt     463_7.txt      6890_8.txt     913_10.t
xt
11390_10.txt   2390_7.txt     4640_10.txt    6891_9.txt     9140_10.
txt
11391_8.txt    2391_10.txt    4641_9.txt     6892_8.txt     9141_8.t
xt
11392_8.txt    2392_8.txt     4642_10.txt    6893_10.txt    9142_10.
txt
11393_7.txt    2393_8.txt     4643_10.txt    6894_7.txt     9143_10.
txt
11394_10.txt   2394_7.txt     4644_10.txt    6895_10.txt    9144_7.t
xt
11395_8.txt    2395_7.txt     4645_9.txt     6896_7.txt     9145_10.
txt
11396_10.txt   2396_7.txt     4646_7.txt     6897_10.txt    9146_9.t
xt
11397_10.txt   2397_9.txt     4647_8.txt     6898_9.txt     9147_10.
txt
11398_7.txt    2398_9.txt     4648_9.txt     6899_9.txt     9148_7.t
xt
11399_8.txt    2399_7.txt     4649_10.txt    689_10.txt     9149_9.t
xt
1139_8.txt     239_7.txt      464_10.txt     68_10.txt      914_8.tx
t
113_10.txt     23_7.txt       4650_7.txt     6900_8.txt     9150_8.t
xt
11400_10.txt   2400_7.txt     4651_7.txt     6901_10.txt    9151_10.
txt
```

```
11401_9.txt   2401_8.txt    4652_9.txt    6902_7.txt    9152_8.t
xt
11402_10.txt  2402_7.txt    4653_10.txt   6903_9.txt    9153_8.t
xt
11403_9.txt   2403_10.txt   4654_7.txt    6904_8.txt    9154_7.t
xt
11404_10.txt  2404_8.txt    4655_7.txt    6905_7.txt    9155_10.
txt
11405_8.txt   2405_10.txt   4656_8.txt    6906_8.txt    9156_10.
txt
11406_9.txt   2406_10.txt   4657_10.txt   6907_8.txt    9157_10.
txt
11407_7.txt   2407_10.txt   4658_8.txt    6908_7.txt    9158_10.
txt
11408_7.txt   2408_8.txt    4659_8.txt    6909_9.txt    9159_10.
txt
11409_9.txt   2409_9.txt    465_10.txt    690_9.txt     915_10.t
xt
1140_10.txt   240_10.txt    4660_7.txt    6910_9.txt    9160_8.t
xt
11410_7.txt   2410_9.txt    4661_10.txt   6911_10.txt   9161_10.
txt
11411_8.txt   2411_9.txt    4662_8.txt    6912_10.txt   9162_10.
txt
11412_10.txt  2412_10.txt   4663_8.txt    6913_9.txt    9163_10.
txt
11413_10.txt  2413_10.txt   4664_8.txt    6914_8.txt    9164_10.
txt
11414_9.txt   2414_10.txt   4665_10.txt   6915_9.txt    9165_10.
txt
11415_8.txt   2415_10.txt   4666_8.txt    6916_8.txt    9166_9.t
xt
11416_10.txt  2416_10.txt   4667_10.txt   6917_9.txt    9167_7.t
xt
11417_7.txt   2417_9.txt    4668_10.txt   6918_10.txt   9168_7.t
xt
11418_7.txt   2418_8.txt    4669_10.txt   6919_9.txt    9169_8.t
xt
11419_10.txt  2419_10.txt   466_8.txt     691_9.txt     916_10.t
xt
```

```
1141_10.txt    241_8.txt      4670_9.txt     6920_8.txt     9170_9.t
xt
11420_9.txt    2420_10.txt    4671_10.txt    6921_9.txt     9171_9.t
xt
11421_10.txt   2421_9.txt     4672_10.txt    6922_8.txt     9172_10.
txt
11422_8.txt    2422_9.txt     4673_9.txt     6923_10.txt    9173_10.
txt
11423_8.txt    2423_7.txt     4674_7.txt     6924_7.txt     9174_9.t
xt
11424_8.txt    2424_7.txt     4675_9.txt     6925_9.txt     9175_10.
txt
11425_8.txt    2425_7.txt     4676_9.txt     6926_10.txt    9176_10.
txt
11426_10.txt   2426_9.txt     4677_9.txt     6927_10.txt    9177_10.
txt
11427_10.txt   2427_10.txt    4678_10.txt    6928_9.txt     9178_10.
txt
11428_7.txt    2428_8.txt     4679_10.txt    6929_8.txt     9179_10.
txt
11429_7.txt    2429_8.txt     467_7.txt      692_8.txt      917_10.t
xt
1142_10.txt    242_8.txt      4680_10.txt    6930_8.txt     9180_9.t
xt
11430_8.txt    2430_10.txt    4681_7.txt     6931_8.txt     9181_10.
txt
11431_10.txt   2431_8.txt     4682_9.txt     6932_7.txt     9182_10.
txt
11432_9.txt    2432_7.txt     4683_7.txt     6933_9.txt     9183_10.
txt
11433_8.txt    2433_8.txt     4684_8.txt     6934_9.txt     9184_10.
txt
11434_7.txt    2434_8.txt     4685_7.txt     6935_8.txt     9185_9.t
xt
11435_10.txt   2435_8.txt     4686_10.txt    6936_7.txt     9186_8.t
xt
11436_9.txt    2436_10.txt    4687_8.txt     6937_8.txt     9187_10.
txt
11437_9.txt    2437_10.txt    4688_10.txt    6938_9.txt     9188_10.
txt
```

```
11438_10.txt  2438_9.txt    4689_10.txt   6939_9.txt    9189_10.
txt
11439_8.txt   2439_8.txt    468_7.txt     693_10.txt    918_10.t
xt
1143_7.txt    243_10.txt    4690_9.txt    6940_10.txt   9190_7.t
xt
11440_10.txt  2440_10.txt   4691_10.txt   6941_9.txt    9191_10.
txt
11441_10.txt  2441_10.txt   4692_10.txt   6942_10.txt   9192_7.t
xt
11442_10.txt  2442_10.txt   4693_7.txt    6943_9.txt    9193_10.
txt
11443_10.txt  2443_9.txt    4694_10.txt   6944_9.txt    9194_9.t
xt
11444_10.txt  2444_8.txt    4695_9.txt    6945_10.txt   9195_10.
txt
11445_7.txt   2445_10.txt   4696_10.txt   6946_10.txt   9196_10.
txt
11446_10.txt  2446_10.txt   4697_7.txt    6947_10.txt   9197_7.t
xt
11447_7.txt   2447_10.txt   4698_10.txt   6948_10.txt   9198_8.t
xt
11448_10.txt  2448_8.txt    4699_8.txt    6949_10.txt   9199_8.t
xt
11449_10.txt  2449_7.txt    469_7.txt     694_9.txt     919_8.tx
t
1144_8.txt    244_10.txt    46_9.txt      6950_8.txt    91_8.txt
11450_10.txt  2450_9.txt    4700_9.txt    6951_10.txt   9200_8.t
xt
11451_8.txt   2451_8.txt    4701_10.txt   6952_7.txt    9201_8.t
xt
11452_8.txt   2452_10.txt   4702_8.txt    6953_7.txt    9202_8.t
xt
11453_9.txt   2453_8.txt    4703_9.txt    6954_8.txt    9203_9.t
xt
11454_10.txt  2454_10.txt   4704_8.txt    6955_10.txt   9204_10.
txt
11455_8.txt   2455_8.txt    4705_7.txt    6956_8.txt    9205_7.t
xt
11456_10.txt  2456_8.txt    4706_8.txt    6957_7.txt    9206_8.t
```

```
xt
11457_10.txt  2457_8.txt    4707_7.txt    6958_7.txt    9207_10.
txt
11458_7.txt   2458_8.txt    4708_7.txt    6959_7.txt    9208_7.t
xt
11459_10.txt  2459_8.txt    4709_9.txt    695_10.txt    9209_7.t
xt
1145_8.txt    245_9.txt     470_10.txt    6960_7.txt    920_10.t
xt
11460_10.txt  2460_10.txt   4710_7.txt    6961_7.txt    9210_10.
txt
11461_10.txt  2461_10.txt   4711_8.txt    6962_7.txt    9211_9.t
xt
11462_10.txt  2462_10.txt   4712_7.txt    6963_8.txt    9212_9.t
xt
11463_9.txt   2463_10.txt   4713_8.txt    6964_7.txt    9213_7.t
xt
11464_10.txt  2464_10.txt   4714_10.txt   6965_7.txt    9214_7.t
xt
11465_7.txt   2465_10.txt   4715_9.txt    6966_8.txt    9215_7.t
xt
11466_10.txt  2466_9.txt    4716_9.txt    6967_7.txt    9216_10.
txt
11467_10.txt  2467_10.txt   4717_8.txt    6968_7.txt    9217_7.t
xt
11468_9.txt   2468_10.txt   4718_7.txt    6969_10.txt   9218_10.
txt
11469_10.txt  2469_10.txt   4719_10.txt   696_10.txt    9219_10.
txt
1146_7.txt    246_7.txt     471_7.txt     6970_8.txt    921_7.tx
t
11470_10.txt  2470_10.txt   4720_8.txt    6971_10.txt   9220_8.t
xt
11471_7.txt   2471_10.txt   4721_8.txt    6972_9.txt    9221_7.t
xt
11472_7.txt   2472_10.txt   4722_8.txt    6973_10.txt   9222_7.t
xt
11473_10.txt  2473_10.txt   4723_8.txt    6974_10.txt   9223_9.t
xt
11474_10.txt  2474_10.txt   4724_7.txt    6975_8.txt    9224_10.
```

```
txt
11475_8.txt    2475_10.txt    4725_9.txt     6976_9.txt     9225_10.
txt
11476_10.txt   2476_10.txt    4726_7.txt     6977_9.txt     9226_8.t
xt
11477_8.txt    2477_7.txt     4727_7.txt     6978_8.txt     9227_9.t
xt
11478_8.txt    2478_10.txt    4728_8.txt     6979_10.txt    9228_10.
txt
11479_10.txt   2479_10.txt    4729_10.txt    697_8.txt      9229_7.t
xt
1147_8.txt     247_10.txt     472_10.txt     6980_9.txt     922_8.tx
t
11480_10.txt   2480_8.txt     4730_10.txt    6981_9.txt     9230_8.t
xt
11481_10.txt   2481_8.txt     4731_8.txt     6982_9.txt     9231_8.t
xt
11482_9.txt    2482_10.txt    4732_10.txt    6983_8.txt     9232_8.t
xt
11483_10.txt   2483_7.txt     4733_9.txt     6984_8.txt     9233_8.t
xt
11484_9.txt    2484_10.txt    4734_9.txt     6985_10.txt    9234_8.t
xt
11485_10.txt   2485_10.txt    4735_9.txt     6986_8.txt     9235_9.t
xt
11486_8.txt    2486_10.txt    4736_10.txt    6987_10.txt    9236_10.
txt
11487_10.txt   2487_10.txt    4737_7.txt     6988_10.txt    9237_7.t
xt
11488_10.txt   2488_10.txt    4738_7.txt     6989_7.txt     9238_10.
txt
11489_10.txt   2489_7.txt     4739_7.txt     698_7.txt      9239_8.t
xt
1148_8.txt     248_10.txt     473_9.txt      6990_8.txt     923_9.tx
t
11490_9.txt    2490_8.txt     4740_8.txt     6991_9.txt     9240_10.
txt
11491_8.txt    2491_8.txt     4741_7.txt     6992_10.txt    9241_10.
txt
11492_7.txt    2492_10.txt    4742_9.txt     6993_7.txt     9242_10.
```

```
txt
11493_7.txt   2493_10.txt   4743_8.txt    6994_8.txt    9243_8.t
xt
11494_7.txt   2494_10.txt   4744_9.txt    6995_7.txt    9244_8.t
xt
11495_10.txt  2495_9.txt    4745_7.txt    6996_8.txt    9245_8.t
xt
11496_10.txt  2496_8.txt    4746_8.txt    6997_7.txt    9246_8.t
xt
11497_10.txt  2497_8.txt    4747_8.txt    6998_7.txt    9247_10.
txt
11498_7.txt   2498_7.txt    4748_7.txt    6999_7.txt    9248_9.t
xt
11499_10.txt  2499_10.txt   4749_8.txt    699_8.txt     9249_9.t
xt
1149_8.txt    249_10.txt    474_7.txt     69_10.txt     924_7.tx
t
114_10.txt    24_8.txt      4750_7.txt    6_10.txt      9250_8.t
xt
11500_10.txt  2500_9.txt    4751_8.txt    7000_7.txt    9251_9.t
xt
11501_8.txt   2501_8.txt    4752_7.txt    7001_10.txt   9252_10.
txt
11502_10.txt  2502_8.txt    4753_10.txt   7002_10.txt   9253_10.
txt
11503_10.txt  2503_10.txt   4754_10.txt   7003_10.txt   9254_7.t
xt
11504_8.txt   2504_10.txt   4755_7.txt    7004_10.txt   9255_10.
txt
11505_7.txt   2505_9.txt    4756_10.txt   7005_9.txt    9256_10.
txt
11506_8.txt   2506_9.txt    4757_8.txt    7006_8.txt    9257_7.t
xt
11507_10.txt  2507_7.txt    4758_8.txt    7007_8.txt    9258_10.
txt
11508_8.txt   2508_10.txt   4759_7.txt    7008_8.txt    9259_10.
txt
11509_8.txt   2509_9.txt    475_10.txt    7009_10.txt   925_10.t
xt
1150_10.txt   250_7.txt     4760_7.txt    700_8.txt     9260_7.t
```

```
xt
11510_10.txt  2510_10.txt   4761_10.txt   7010_10.txt   9261_7.t
xt
11511_9.txt   2511_10.txt   4762_10.txt   7011_7.txt    9262_8.t
xt
11512_10.txt  2512_10.txt   4763_8.txt    7012_10.txt   9263_8.t
xt
11513_8.txt   2513_9.txt    4764_10.txt   7013_9.txt    9264_9.t
xt
11514_7.txt   2514_8.txt    4765_7.txt    7014_10.txt   9265_9.t
xt
11515_10.txt  2515_10.txt   4766_10.txt   7015_8.txt    9266_9.t
xt
11516_8.txt   2516_9.txt    4767_9.txt    7016_7.txt    9267_7.t
xt
11517_9.txt   2517_9.txt    4768_10.txt   7017_9.txt    9268_9.t
xt
11518_8.txt   2518_10.txt   4769_7.txt    7018_10.txt   9269_7.t
xt
11519_9.txt   2519_10.txt   476_7.txt     7019_7.txt    926_7.tx
t
1151_9.txt    251_10.txt    4770_10.txt   701_7.txt     9270_9.t
xt
11520_8.txt   2520_10.txt   4771_10.txt   7020_10.txt   9271_8.t
xt
11521_10.txt  2521_10.txt   4772_10.txt   7021_10.txt   9272_7.t
xt
11522_8.txt   2522_8.txt    4773_10.txt   7022_7.txt    9273_7.t
xt
11523_7.txt   2523_9.txt    4774_10.txt   7023_9.txt    9274_8.t
xt
11524_7.txt   2524_10.txt   4775_7.txt    7024_8.txt    9275_9.t
xt
11525_8.txt   2525_8.txt    4776_8.txt    7025_8.txt    9276_10.
txt
11526_10.txt  2526_9.txt    4777_10.txt   7026_7.txt    9277_9.t
xt
11527_7.txt   2527_9.txt    4778_9.txt    7027_7.txt    9278_7.t
xt
11528_7.txt   2528_10.txt   4779_10.txt   7028_9.txt    9279_7.t
```

```
xt
11529_7.txt   2529_7.txt    477_10.txt    7029_7.txt    927_10.t
xt
1152_10.txt   252_9.txt     4780_9.txt    702_10.txt    9280_9.t
xt
11530_8.txt   2530_8.txt    4781_8.txt    7030_10.txt   9281_10.
txt
11531_10.txt  2531_7.txt    4782_9.txt    7031_10.txt   9282_8.t
xt
11532_8.txt   2532_8.txt    4783_10.txt   7032_10.txt   9283_8.t
xt
11533_8.txt   2533_7.txt    4784_10.txt   7033_7.txt    9284_8.t
xt
11534_7.txt   2534_8.txt    4785_10.txt   7034_10.txt   9285_10.
txt
11535_7.txt   2535_7.txt    4786_10.txt   7035_8.txt    9286_9.t
xt
11536_7.txt   2536_7.txt    4787_10.txt   7036_10.txt   9287_9.t
xt
11537_7.txt   2537_7.txt    4788_10.txt   7037_10.txt   9288_8.t
xt
11538_7.txt   2538_10.txt   4789_10.txt   7038_9.txt    9289_7.t
xt
11539_9.txt   2539_10.txt   478_7.txt     7039_7.txt    928_10.t
xt
1153_10.txt   253_7.txt     4790_10.txt   703_10.txt    9290_9.t
xt
11540_10.txt  2540_8.txt    4791_10.txt   7040_10.txt   9291_7.t
xt
11541_10.txt  2541_7.txt    4792_10.txt   7041_7.txt    9292_10.
txt
11542_7.txt   2542_10.txt   4793_7.txt    7042_10.txt   9293_7.t
xt
11543_8.txt   2543_10.txt   4794_8.txt    7043_7.txt    9294_7.t
xt
11544_10.txt  2544_8.txt    4795_9.txt    7044_10.txt   9295_10.
txt
11545_8.txt   2545_8.txt    4796_8.txt    7045_10.txt   9296_10.
txt
11546_9.txt   2546_10.txt   4797_10.txt   7046_7.txt    9297_10.
```

```
txt
11547_9.txt    2547_10.txt    4798_8.txt    7047_10.txt    9298_9.t
xt
11548_10.txt   2548_7.txt     4799_10.txt   7048_10.txt    9299_8.t
xt
11549_7.txt    2549_8.txt     479_10.txt    7049_7.txt     929_10.t
xt
1154_10.txt    254_8.txt      47_8.txt      704_10.txt     92_9.txt
11550_7.txt    2550_9.txt     4800_7.txt    7050_7.txt     9300_10.
txt
11551_7.txt    2551_9.txt     4801_7.txt    7051_10.txt    9301_10.
txt
11552_9.txt    2552_10.txt    4802_8.txt    7052_8.txt     9302_10.
txt
11553_9.txt    2553_7.txt     4803_8.txt    7053_7.txt     9303_8.t
xt
11554_8.txt    2554_7.txt     4804_7.txt    7054_9.txt     9304_9.t
xt
11555_10.txt   2555_10.txt    4805_8.txt    7055_8.txt     9305_10.
txt
11556_7.txt    2556_8.txt     4806_8.txt    7056_7.txt     9306_9.t
xt
11557_8.txt    2557_10.txt    4807_10.txt   7057_9.txt     9307_10.
txt
11558_10.txt   2558_10.txt    4808_8.txt    7058_8.txt     9308_7.t
xt
11559_10.txt   2559_9.txt     4809_10.txt   7059_10.txt    9309_9.t
xt
1155_10.txt    255_10.txt     480_10.txt    705_10.txt     930_7.tx
t
11560_9.txt    2560_7.txt     4810_9.txt    7060_10.txt    9310_10.
txt
11561_10.txt   2561_7.txt     4811_8.txt    7061_7.txt     9311_10.
txt
11562_9.txt    2562_10.txt    4812_8.txt    7062_7.txt     9312_10.
txt
11563_10.txt   2563_10.txt    4813_9.txt    7063_8.txt     9313_9.t
xt
11564_9.txt    2564_10.txt    4814_10.txt   7064_8.txt     9314_10.
txt
```

```
11565_9.txt   2565_9.txt    4815_10.txt   7065_7.txt    9315_7.t
xt
11566_10.txt  2566_8.txt    4816_8.txt    7066_8.txt    9316_7.t
xt
11567_8.txt   2567_9.txt    4817_10.txt   7067_7.txt    9317_8.t
xt
11568_7.txt   2568_10.txt   4818_9.txt    7068_8.txt    9318_9.t
xt
11569_8.txt   2569_10.txt   4819_7.txt    7069_8.txt    9319_8.t
xt
1156_9.txt    256_9.txt     481_10.txt    706_10.txt    931_10.t
xt
11570_9.txt   2570_10.txt   4820_7.txt    7070_9.txt    9320_10.
txt
11571_9.txt   2571_7.txt    4821_10.txt   7071_9.txt    9321_8.t
xt
11572_8.txt   2572_10.txt   4822_9.txt    7072_9.txt    9322_8.t
xt
11573_10.txt  2573_10.txt   4823_7.txt    7073_10.txt   9323_10.
txt
11574_10.txt  2574_7.txt    4824_7.txt    7074_7.txt    9324_7.t
xt
11575_10.txt  2575_7.txt    4825_8.txt    7075_10.txt   9325_10.
txt
11576_7.txt   2576_8.txt    4826_7.txt    7076_10.txt   9326_10.
txt
11577_8.txt   2577_10.txt   4827_9.txt    7077_10.txt   9327_8.t
xt
11578_10.txt  2578_10.txt   4828_7.txt    7078_9.txt    9328_7.t
xt
11579_10.txt  2579_8.txt    4829_8.txt    7079_9.txt    9329_8.t
xt
1157_10.txt   257_7.txt     482_8.txt     707_8.txt     932_10.t
xt
11580_10.txt  2580_10.txt   4830_9.txt    7080_10.txt   9330_8.t
xt
11581_10.txt  2581_8.txt    4831_8.txt    7081_8.txt    9331_9.t
xt
11582_10.txt  2582_8.txt    4832_8.txt    7082_8.txt    9332_10.
txt
```

```
11583_10.txt  2583_10.txt   4833_10.txt   7083_10.txt   9333_8.t
xt
11584_8.txt   2584_7.txt    4834_8.txt    7084_10.txt   9334_9.t
xt
11585_10.txt  2585_7.txt    4835_10.txt   7085_9.txt    9335_10.
txt
11586_9.txt   2586_10.txt   4836_8.txt    7086_10.txt   9336_9.t
xt
11587_10.txt  2587_9.txt    4837_10.txt   7087_9.txt    9337_10.
txt
11588_10.txt  2588_7.txt    4838_10.txt   7088_10.txt   9338_8.t
xt
11589_10.txt  2589_7.txt    4839_10.txt   7089_9.txt    9339_10.
txt
1158_9.txt    258_7.txt     483_8.txt     708_8.txt     933_10.t
xt
11590_10.txt  2590_7.txt    4840_7.txt    7090_10.txt   9340_8.t
xt
11591_10.txt  2591_7.txt    4841_7.txt    7091_9.txt    9341_7.t
xt
11592_10.txt  2592_10.txt   4842_10.txt   7092_8.txt    9342_8.t
xt
11593_10.txt  2593_10.txt   4843_7.txt    7093_10.txt   9343_8.t
xt
11594_10.txt  2594_9.txt    4844_8.txt    7094_10.txt   9344_8.t
xt
11595_10.txt  2595_9.txt    4845_9.txt    7095_9.txt    9345_9.t
xt
11596_10.txt  2596_10.txt   4846_7.txt    7096_10.txt   9346_10.
txt
11597_10.txt  2597_8.txt    4847_9.txt    7097_9.txt    9347_7.t
xt
11598_8.txt   2598_10.txt   4848_7.txt    7098_9.txt    9348_7.t
xt
11599_8.txt   2599_10.txt   4849_10.txt   7099_10.txt   9349_8.t
xt
1159_10.txt   259_8.txt     484_8.txt     709_10.txt    934_9.tx
t
115_10.txt    25_7.txt      4850_10.txt   70_9.txt      9350_10.
txt
```

```
11600_7.txt    2600_10.txt    4851_7.txt     7100_8.txt     9351_8.t
xt
11601_8.txt    2601_10.txt    4852_8.txt     7101_10.txt    9352_10.
txt
11602_8.txt    2602_10.txt    4853_10.txt    7102_7.txt     9353_8.t
xt
11603_10.txt   2603_8.txt     4854_7.txt     7103_7.txt     9354_9.t
xt
11604_8.txt    2604_8.txt     4855_7.txt     7104_7.txt     9355_7.t
xt
11605_10.txt   2605_7.txt     4856_10.txt    7105_9.txt     9356_9.t
xt
11606_10.txt   2606_8.txt     4857_7.txt     7106_8.txt     9357_7.t
xt
11607_10.txt   2607_9.txt     4858_8.txt     7107_10.txt    9358_8.t
xt
11608_10.txt   2608_10.txt    4859_7.txt     7108_9.txt     9359_8.t
xt
11609_10.txt   2609_10.txt    485_8.txt      7109_10.txt    935_9.tx
t
1160_10.txt    260_7.txt      4860_10.txt    710_9.txt      9360_10.
txt
11610_10.txt   2610_9.txt     4861_8.txt     7110_8.txt     9361_9.t
xt
11611_9.txt    2611_9.txt     4862_7.txt     7111_8.txt     9362_7.t
xt
11612_10.txt   2612_10.txt    4863_10.txt    7112_9.txt     9363_10.
txt
11613_7.txt    2613_9.txt     4864_7.txt     7113_8.txt     9364_9.t
xt
11614_7.txt    2614_9.txt     4865_7.txt     7114_9.txt     9365_9.t
xt
11615_9.txt    2615_9.txt     4866_7.txt     7115_9.txt     9366_9.t
xt
11616_8.txt    2616_10.txt    4867_9.txt     7116_9.txt     9367_9.t
xt
11617_9.txt    2617_10.txt    4868_8.txt     7117_10.txt    9368_8.t
xt
11618_8.txt    2618_7.txt     4869_10.txt    7118_10.txt    9369_10.
txt
```

```
11619_9.txt    2619_8.txt     486_9.txt      7119_10.txt    936_8.tx
t
1161_8.txt     261_8.txt      4870_10.txt    711_10.txt     9370_7.t
xt
11620_8.txt    2620_10.txt    4871_8.txt     7120_9.txt     9371_7.t
xt
11621_9.txt    2621_8.txt     4872_8.txt     7121_10.txt    9372_9.t
xt
11622_10.txt   2622_8.txt     4873_8.txt     7122_8.txt     9373_9.t
xt
11623_8.txt    2623_7.txt     4874_10.txt    7123_10.txt    9374_9.t
xt
11624_8.txt    2624_7.txt     4875_7.txt     7124_8.txt     9375_10.
txt
11625_7.txt    2625_8.txt     4876_10.txt    7125_10.txt    9376_10.
txt
11626_9.txt    2626_8.txt     4877_9.txt     7126_10.txt    9377_7.t
xt
11627_8.txt    2627_9.txt     4878_10.txt    7127_10.txt    9378_9.t
xt
11628_10.txt   2628_10.txt    4879_10.txt    7128_7.txt     9379_7.t
xt
11629_10.txt   2629_9.txt     487_8.txt      7129_10.txt    937_10.t
xt
1162_9.txt     262_8.txt      4880_9.txt     712_9.txt      9380_9.t
xt
11630_8.txt    2630_8.txt     4881_9.txt     7130_10.txt    9381_8.t
xt
11631_7.txt    2631_7.txt     4882_9.txt     7131_9.txt     9382_9.t
xt
11632_7.txt    2632_7.txt     4883_10.txt    7132_10.txt    9383_8.t
xt
11633_8.txt    2633_8.txt     4884_10.txt    7133_10.txt    9384_8.t
xt
11634_9.txt    2634_7.txt     4885_9.txt     7134_10.txt    9385_8.t
xt
11635_8.txt    2635_7.txt     4886_8.txt     7135_10.txt    9386_8.t
xt
11636_8.txt    2636_9.txt     4887_10.txt    7136_10.txt    9387_9.t
xt
```

```
11637_8.txt    2637_7.txt    4888_8.txt    7137_8.txt    9388_8.t
xt
11638_7.txt    2638_10.txt   4889_10.txt   7138_10.txt   9389_7.t
xt
11639_10.txt   2639_7.txt    488_9.txt     7139_10.txt   938_9.tx
t
1163_7.txt     263_9.txt     4890_10.txt   713_10.txt    9390_9.t
xt
11640_9.txt    2640_10.txt   4891_10.txt   7140_10.txt   9391_8.t
xt
11641_7.txt    2641_10.txt   4892_10.txt   7141_8.txt    9392_9.t
xt
11642_10.txt   2642_10.txt   4893_10.txt   7142_8.txt    9393_10.
txt
11643_7.txt    2643_10.txt   4894_10.txt   7143_10.txt   9394_10.
txt
11644_10.txt   2644_8.txt    4895_10.txt   7144_10.txt   9395_9.t
xt
11645_8.txt    2645_8.txt    4896_7.txt    7145_9.txt    9396_9.t
xt
11646_10.txt   2646_10.txt   4897_8.txt    7146_10.txt   9397_9.t
xt
11647_8.txt    2647_10.txt   4898_7.txt    7147_10.txt   9398_9.t
xt
11648_9.txt    2648_8.txt    4899_10.txt   7148_10.txt   9399_7.t
xt
11649_10.txt   2649_7.txt    489_7.txt     7149_10.txt   939_10.t
xt
1164_10.txt    264_7.txt     48_7.txt      714_10.txt    93_10.tx
t
11650_9.txt    2650_10.txt   4900_8.txt    7150_9.txt    9400_8.t
xt
11651_10.txt   2651_10.txt   4901_9.txt    7151_10.txt   9401_10.
txt
11652_10.txt   2652_10.txt   4902_8.txt    7152_9.txt    9402_7.t
xt
11653_10.txt   2653_10.txt   4903_10.txt   7153_8.txt    9403_8.t
xt
11654_9.txt    2654_10.txt   4904_10.txt   7154_10.txt   9404_10.
txt
```

```
11655_10.txt  2655_10.txt   4905_10.txt   7155_8.txt    9405_10.
txt
11656_9.txt   2656_10.txt   4906_9.txt    7156_10.txt   9406_8.t
xt
11657_9.txt   2657_10.txt   4907_8.txt    7157_7.txt    9407_8.t
xt
11658_10.txt  2658_10.txt   4908_10.txt   7158_8.txt    9408_10.
txt
11659_9.txt   2659_8.txt    4909_8.txt    7159_10.txt   9409_8.t
xt
1165_7.txt    265_7.txt     490_9.txt     715_10.txt    940_10.t
xt
11660_7.txt   2660_10.txt   4910_8.txt    7160_8.txt    9410_7.t
xt
11661_7.txt   2661_10.txt   4911_9.txt    7161_9.txt    9411_8.t
xt
11662_9.txt   2662_10.txt   4912_8.txt    7162_10.txt   9412_10.
txt
11663_8.txt   2663_10.txt   4913_9.txt    7163_10.txt   9413_9.t
xt
11664_9.txt   2664_9.txt    4914_7.txt    7164_10.txt   9414_9.t
xt
11665_8.txt   2665_10.txt   4915_7.txt    7165_10.txt   9415_8.t
xt
11666_10.txt  2666_10.txt   4916_9.txt    7166_9.txt    9416_10.
txt
11667_9.txt   2667_8.txt    4917_8.txt    7167_9.txt    9417_10.
txt
11668_7.txt   2668_9.txt    4918_7.txt    7168_10.txt   9418_10.
txt
11669_9.txt   2669_10.txt   4919_7.txt    7169_8.txt    9419_8.t
xt
1166_8.txt    266_7.txt     491_7.txt     716_10.txt    941_10.t
xt
11670_7.txt   2670_10.txt   4920_7.txt    7170_9.txt    9420_8.t
xt
11671_8.txt   2671_7.txt    4921_8.txt    7171_10.txt   9421_10.
txt
11672_10.txt  2672_7.txt    4922_7.txt    7172_10.txt   9422_9.t
xt
```

```
11673_8.txt   2673_8.txt   4923_7.txt   7173_10.txt   9423_8.t
xt
11674_10.txt  2674_7.txt   4924_7.txt   7174_9.txt    9424_10.
txt
11675_10.txt  2675_9.txt   4925_7.txt   7175_10.txt   9425_10.
txt
11676_8.txt   2676_10.txt  4926_8.txt   7176_10.txt   9426_10.
txt
11677_8.txt   2677_9.txt   4927_8.txt   7177_8.txt    9427_10.
txt
11678_8.txt   2678_8.txt   4928_7.txt   7178_10.txt   9428_10.
txt
11679_7.txt   2679_8.txt   4929_7.txt   7179_8.txt    9429_9.t
xt
1167_7.txt    267_7.txt    492_7.txt    717_7.txt     942_10.t
xt
11680_8.txt   2680_7.txt   4930_7.txt   7180_9.txt    9430_9.t
xt
11681_9.txt   2681_7.txt   4931_8.txt   7181_10.txt   9431_10.
txt
11682_7.txt   2682_9.txt   4932_8.txt   7182_10.txt   9432_9.t
xt
11683_8.txt   2683_10.txt  4933_8.txt   7183_7.txt    9433_10.
txt
11684_8.txt   2684_10.txt  4934_8.txt   7184_7.txt    9434_10.
txt
11685_8.txt   2685_10.txt  4935_7.txt   7185_10.txt   9435_7.t
xt
11686_10.txt  2686_9.txt   4936_8.txt   7186_7.txt    9436_10.
txt
11687_10.txt  2687_10.txt  4937_7.txt   7187_8.txt    9437_9.t
xt
11688_10.txt  2688_8.txt   4938_7.txt   7188_8.txt    9438_10.
txt
11689_8.txt   2689_9.txt   4939_7.txt   7189_10.txt   9439_8.t
xt
1168_8.txt    268_8.txt    493_10.txt   718_10.txt    943_10.t
xt
11690_9.txt   2690_9.txt   4940_7.txt   7190_9.txt    9440_8.t
xt
```

```
11691_8.txt    2691_8.txt     4941_10.txt    7191_9.txt     9441_9.t
xt
11692_7.txt    2692_8.txt     4942_7.txt     7192_8.txt     9442_10.
txt
11693_7.txt    2693_7.txt     4943_10.txt    7193_10.txt    9443_9.t
xt
11694_7.txt    2694_10.txt    4944_7.txt     7194_7.txt     9444_10.
txt
11695_9.txt    2695_10.txt    4945_7.txt     7195_10.txt    9445_10.
txt
11696_7.txt    2696_8.txt     4946_10.txt    7196_8.txt     9446_10.
txt
11697_10.txt   2697_9.txt     4947_8.txt     7197_7.txt     9447_8.t
xt
11698_10.txt   2698_8.txt     4948_7.txt     7198_9.txt     9448_7.t
xt
11699_10.txt   2699_10.txt    4949_8.txt     7199_8.txt     9449_10.
txt
1169_8.txt     269_8.txt      494_9.txt      719_10.txt     944_10.t
xt
116_10.txt     26_9.txt       4950_8.txt     71_10.txt      9450_10.
txt
11700_10.txt   2700_10.txt    4951_8.txt     7200_9.txt     9451_8.t
xt
11701_9.txt    2701_9.txt     4952_7.txt     7201_8.txt     9452_9.t
xt
11702_10.txt   2702_9.txt     4953_8.txt     7202_8.txt     9453_8.t
xt
11703_9.txt    2703_9.txt     4954_10.txt    7203_9.txt     9454_10.
txt
11704_10.txt   2704_10.txt    4955_9.txt     7204_8.txt     9455_9.t
xt
11705_7.txt    2705_10.txt    4956_10.txt    7205_10.txt    9456_8.t
xt
11706_10.txt   2706_9.txt     4957_8.txt     7206_9.txt     9457_9.t
xt
11707_10.txt   2707_10.txt    4958_10.txt    7207_7.txt     9458_10.
txt
11708_8.txt    2708_7.txt     4959_7.txt     7208_7.txt     9459_8.t
xt
```

```
11709_10.txt  2709_8.txt    495_7.txt     7209_8.txt    945_10.t
xt
1170_8.txt    270_10.txt    4960_9.txt    720_7.txt     9460_8.t
xt
11710_10.txt  2710_7.txt    4961_10.txt   7210_8.txt    9461_7.t
xt
11711_10.txt  2711_7.txt    4962_7.txt    7211_10.txt   9462_9.t
xt
11712_10.txt  2712_10.txt   4963_10.txt   7212_9.txt    9463_10.
txt
11713_10.txt  2713_8.txt    4964_10.txt   7213_10.txt   9464_8.t
xt
11714_10.txt  2714_10.txt   4965_9.txt    7214_7.txt    9465_7.t
xt
11715_7.txt   2715_8.txt    4966_10.txt   7215_7.txt    9466_7.t
xt
11716_7.txt   2716_10.txt   4967_10.txt   7216_10.txt   9467_10.
txt
11717_8.txt   2717_10.txt   4968_10.txt   7217_7.txt    9468_7.t
xt
11718_10.txt  2718_9.txt    4969_7.txt    7218_9.txt    9469_10.
txt
11719_7.txt   2719_10.txt   496_10.txt    7219_7.txt    946_8.tx
t
1171_10.txt   271_10.txt    4970_8.txt    721_10.txt    9470_8.t
xt
11720_8.txt   2720_7.txt    4971_7.txt    7220_7.txt    9471_7.t
xt
11721_7.txt   2721_9.txt    4972_9.txt    7221_8.txt    9472_8.t
xt
11722_8.txt   2722_8.txt    4973_9.txt    7222_8.txt    9473_10.
txt
11723_7.txt   2723_7.txt    4974_10.txt   7223_10.txt   9474_10.
txt
11724_8.txt   2724_9.txt    4975_8.txt    7224_8.txt    9475_9.t
xt
11725_10.txt  2725_8.txt    4976_10.txt   7225_10.txt   9476_9.t
xt
11726_9.txt   2726_9.txt    4977_7.txt    7226_8.txt    9477_9.t
xt
```

```
11727_7.txt   2727_9.txt    4978_7.txt    7227_7.txt    9478_8.t
xt
11728_9.txt   2728_10.txt   4979_10.txt   7228_9.txt    9479_10.
txt
11729_10.txt  2729_10.txt   497_10.txt    7229_7.txt    947_10.t
xt
1172_8.txt    272_10.txt    4980_8.txt    722_7.txt     9480_10.
txt
11730_10.txt  2730_9.txt    4981_10.txt   7230_8.txt    9481_9.t
xt
11731_10.txt  2731_10.txt   4982_7.txt    7231_8.txt    9482_7.t
xt
11732_7.txt   2732_8.txt    4983_7.txt    7232_8.txt    9483_10.
txt
11733_10.txt  2733_10.txt   4984_8.txt    7233_7.txt    9484_10.
txt
11734_10.txt  2734_9.txt    4985_10.txt   7234_8.txt    9485_10.
txt
11735_10.txt  2735_10.txt   4986_8.txt    7235_8.txt    9486_10.
txt
11736_9.txt   2736_9.txt    4987_10.txt   7236_8.txt    9487_8.t
xt
11737_8.txt   2737_9.txt    4988_7.txt    7237_10.txt   9488_10.
txt
11738_9.txt   2738_10.txt   4989_8.txt    7238_9.txt    9489_9.t
xt
11739_8.txt   2739_9.txt    498_10.txt    7239_7.txt    948_10.t
xt
1173_8.txt    273_9.txt     4990_8.txt    723_8.txt     9490_10.
txt
11740_10.txt  2740_8.txt    4991_10.txt   7240_8.txt    9491_10.
txt
11741_10.txt  2741_10.txt   4992_7.txt    7241_8.txt    9492_7.t
xt
11742_9.txt   2742_9.txt    4993_10.txt   7242_7.txt    9493_8.t
xt
11743_7.txt   2743_9.txt    4994_9.txt    7243_10.txt   9494_10.
txt
11744_9.txt   2744_10.txt   4995_9.txt    7244_10.txt   9495_8.t
xt
```

```
11745_10.txt  2745_10.txt  4996_9.txt   7245_10.txt  9496_10.
txt
11746_9.txt   2746_10.txt  4997_9.txt   7246_10.txt  9497_8.t
xt
11747_8.txt   2747_9.txt   4998_9.txt   7247_10.txt  9498_8.t
xt
11748_8.txt   2748_8.txt   4999_9.txt   7248_9.txt   9499_7.t
xt
11749_10.txt  2749_10.txt  499_8.txt    7249_8.txt   949_8.tx
t
1174_10.txt   274_7.txt    49_10.txt    724_10.txt   94_10.tx
t
11750_10.txt  2750_10.txt  4_8.txt      7250_8.txt   9500_7.t
xt
11751_10.txt  2751_10.txt  5000_10.txt  7251_10.txt  9501_10.
txt
11752_10.txt  2752_7.txt   5001_7.txt   7252_8.txt   9502_8.t
xt
11753_10.txt  2753_10.txt  5002_8.txt   7253_10.txt  9503_7.t
xt
11754_7.txt   2754_10.txt  5003_10.txt  7254_10.txt  9504_10.
txt
11755_7.txt   2755_10.txt  5004_9.txt   7255_9.txt   9505_10.
txt
11756_7.txt   2756_9.txt   5005_8.txt   7256_9.txt   9506_8.t
xt
11757_9.txt   2757_10.txt  5006_10.txt  7257_10.txt  9507_10.
txt
11758_7.txt   2758_7.txt   5007_10.txt  7258_10.txt  9508_8.t
xt
11759_10.txt  2759_10.txt  5008_10.txt  7259_7.txt   9509_9.t
xt
1175_9.txt    275_10.txt   5009_9.txt   725_10.txt   950_9.tx
t
11760_8.txt   2760_8.txt   500_9.txt    7260_10.txt  9510_8.t
xt
11761_10.txt  2761_10.txt  5010_10.txt  7261_8.txt   9511_7.t
xt
11762_8.txt   2762_9.txt   5011_10.txt  7262_10.txt  9512_8.t
xt
```

```
11763_7.txt   2763_8.txt    5012_10.txt   7263_10.txt   9513_10.
txt
11764_10.txt  2764_10.txt   5013_10.txt   7264_8.txt    9514_10.
txt
11765_8.txt   2765_9.txt    5014_9.txt    7265_8.txt    9515_10.
txt
11766_8.txt   2766_9.txt    5015_7.txt    7266_7.txt    9516_8.t
xt
11767_8.txt   2767_8.txt    5016_10.txt   7267_7.txt    9517_10.
txt
11768_7.txt   2768_8.txt    5017_8.txt    7268_9.txt    9518_10.
txt
11769_7.txt   2769_10.txt   5018_8.txt    7269_7.txt    9519_10.
txt
1176_10.txt   276_10.txt    5019_10.txt   726_7.txt     951_10.t
xt
11770_10.txt  2770_8.txt    501_10.txt    7270_8.txt    9520_10.
txt
11771_10.txt  2771_7.txt    5020_10.txt   7271_7.txt    9521_10.
txt
11772_10.txt  2772_7.txt    5021_10.txt   7272_10.txt   9522_10.
txt
11773_8.txt   2773_7.txt    5022_8.txt    7273_8.txt    9523_10.
txt
11774_10.txt  2774_10.txt   5023_10.txt   7274_10.txt   9524_9.t
xt
11775_10.txt  2775_7.txt    5024_8.txt    7275_8.txt    9525_8.t
xt
11776_9.txt   2776_8.txt    5025_10.txt   7276_8.txt    9526_10.
txt
11777_9.txt   2777_7.txt    5026_8.txt    7277_7.txt    9527_10.
txt
11778_10.txt  2778_8.txt    5027_10.txt   7278_7.txt    9528_9.t
xt
11779_7.txt   2779_7.txt    5028_10.txt   7279_9.txt    9529_9.t
xt
1177_9.txt    277_8.txt     5029_8.txt    727_9.txt     952_10.t
xt
11780_8.txt   2780_9.txt    502_10.txt    7280_10.txt   9530_9.t
xt
```

```
11781_8.txt   2781_8.txt    5030_9.txt    7281_10.txt   9531_7.t
xt
11782_8.txt   2782_10.txt   5031_10.txt   7282_9.txt    9532_10.
txt
11783_10.txt  2783_8.txt    5032_10.txt   7283_8.txt    9533_9.t
xt
11784_10.txt  2784_8.txt    5033_10.txt   7284_8.txt    9534_10.
txt
11785_10.txt  2785_7.txt    5034_7.txt    7285_9.txt    9535_10.
txt
11786_8.txt   2786_7.txt    5035_7.txt    7286_8.txt    9536_7.t
xt
11787_9.txt   2787_8.txt    5036_9.txt    7287_7.txt    9537_8.t
xt
11788_8.txt   2788_9.txt    5037_10.txt   7288_7.txt    9538_10.
txt
11789_10.txt  2789_7.txt    5038_10.txt   7289_10.txt   9539_7.t
xt
1178_10.txt   278_9.txt     5039_10.txt   728_10.txt    953_10.t
xt
11790_8.txt   2790_10.txt   503_10.txt    7290_10.txt   9540_8.t
xt
11791_8.txt   2791_10.txt   5040_9.txt    7291_10.txt   9541_8.t
xt
11792_8.txt   2792_10.txt   5041_8.txt    7292_10.txt   9542_10.
txt
11793_8.txt   2793_9.txt    5042_9.txt    7293_10.txt   9543_7.t
xt
11794_7.txt   2794_8.txt    5043_7.txt    7294_9.txt    9544_10.
txt
11795_10.txt  2795_7.txt    5044_9.txt    7295_10.txt   9545_7.t
xt
11796_9.txt   2796_10.txt   5045_8.txt    7296_10.txt   9546_8.t
xt
11797_8.txt   2797_10.txt   5046_10.txt   7297_10.txt   9547_9.t
xt
11798_10.txt  2798_8.txt    5047_8.txt    7298_7.txt    9548_10.
txt
11799_8.txt   2799_10.txt   5048_9.txt    7299_8.txt    9549_8.t
xt
```

```
1179_9.txt    279_9.txt     5049_9.txt    729_10.txt    954_10.t
xt
117_10.txt    27_10.txt     504_8.txt     72_7.txt      9550_7.t
xt
11800_8.txt   2800_10.txt   5050_7.txt    7300_8.txt    9551_10.
txt
11801_8.txt   2801_9.txt    5051_7.txt    7301_7.txt    9552_8.t
xt
11802_7.txt   2802_10.txt   5052_10.txt   7302_10.txt   9553_9.t
xt
11803_7.txt   2803_10.txt   5053_8.txt    7303_10.txt   9554_8.t
xt
11804_10.txt  2804_8.txt    5054_7.txt    7304_8.txt    9555_10.
txt
11805_10.txt  2805_9.txt    5055_8.txt    7305_10.txt   9556_10.
txt
11806_10.txt  2806_10.txt   5056_8.txt    7306_10.txt   9557_7.t
xt
11807_7.txt   2807_7.txt    5057_10.txt   7307_10.txt   9558_10.
txt
11808_9.txt   2808_10.txt   5058_10.txt   7308_8.txt    9559_8.t
xt
11809_10.txt  2809_8.txt    5059_7.txt    7309_9.txt    955_7.tx
t
1180_9.txt    280_8.txt     505_9.txt     730_7.txt     9560_9.t
xt
11810_7.txt   2810_10.txt   5060_8.txt    7310_9.txt    9561_8.t
xt
11811_8.txt   2811_10.txt   5061_8.txt    7311_9.txt    9562_8.t
xt
11812_10.txt  2812_8.txt    5062_9.txt    7312_10.txt   9563_9.t
xt
11813_8.txt   2813_8.txt    5063_7.txt    7313_10.txt   9564_10.
txt
11814_7.txt   2814_10.txt   5064_8.txt    7314_10.txt   9565_8.t
xt
11815_10.txt  2815_10.txt   5065_7.txt    7315_10.txt   9566_7.t
xt
11816_8.txt   2816_7.txt    5066_8.txt    7316_10.txt   9567_7.t
xt
```

```
11817_10.txt  2817_9.txt    5067_10.txt   7317_10.txt   9568_7.t
xt
11818_10.txt  2818_7.txt    5068_9.txt    7318_10.txt   9569_9.t
xt
11819_8.txt   2819_10.txt   5069_7.txt    7319_10.txt   956_9.tx
t
1181_9.txt    281_10.txt    506_7.txt     731_9.txt     9570_8.t
xt
11820_8.txt   2820_7.txt    5070_9.txt    7320_10.txt   9571_9.t
xt
11821_10.txt  2821_9.txt    5071_10.txt   7321_10.txt   9572_8.t
xt
11822_10.txt  2822_7.txt    5072_8.txt    7322_10.txt   9573_8.t
xt
11823_9.txt   2823_7.txt    5073_7.txt    7323_9.txt    9574_9.t
xt
11824_10.txt  2824_8.txt    5074_10.txt   7324_10.txt   9575_9.t
xt
11825_8.txt   2825_9.txt    5075_7.txt    7325_9.txt    9576_9.t
xt
11826_10.txt  2826_10.txt   5076_10.txt   7326_10.txt   9577_8.t
xt
11827_9.txt   2827_10.txt   5077_9.txt    7327_10.txt   9578_10.
txt
11828_8.txt   2828_9.txt    5078_9.txt    7328_8.txt    9579_8.t
xt
11829_10.txt  2829_10.txt   5079_8.txt    7329_8.txt    957_10.t
xt
1182_8.txt    282_9.txt     507_10.txt    732_7.txt     9580_7.t
xt
11830_9.txt   2830_9.txt    5080_10.txt   7330_8.txt    9581_10.
txt
11831_7.txt   2831_7.txt    5081_8.txt    7331_10.txt   9582_10.
txt
11832_7.txt   2832_8.txt    5082_7.txt    7332_7.txt    9583_8.t
xt
11833_7.txt   2833_7.txt    5083_8.txt    7333_10.txt   9584_10.
txt
11834_7.txt   2834_9.txt    5084_7.txt    7334_7.txt    9585_10.
txt
```

```
11835_8.txt    2835_7.txt    5085_7.txt    7335_7.txt    9586_10.
txt
11836_10.txt   2836_7.txt    5086_7.txt    7336_10.txt   9587_10.
txt
11837_7.txt    2837_8.txt    5087_7.txt    7337_10.txt   9588_10.
txt
11838_9.txt    2838_10.txt   5088_8.txt    7338_7.txt    9589_10.
txt
11839_9.txt    2839_10.txt   5089_9.txt    7339_10.txt   958_9.tx
t
1183_8.txt     283_8.txt     508_9.txt     733_9.txt     9590_10.
txt
11840_10.txt   2840_9.txt    5090_10.txt   7340_8.txt    9591_10.
txt
11841_9.txt    2841_10.txt   5091_8.txt    7341_8.txt    9592_8.t
xt
11842_10.txt   2842_8.txt    5092_8.txt    7342_7.txt    9593_10.
txt
11843_10.txt   2843_7.txt    5093_10.txt   7343_10.txt   9594_10.
txt
11844_10.txt   2844_9.txt    5094_8.txt    7344_10.txt   9595_10.
txt
11845_10.txt   2845_8.txt    5095_7.txt    7345_7.txt    9596_10.
txt
11846_10.txt   2846_10.txt   5096_7.txt    7346_8.txt    9597_10.
txt
11847_9.txt    2847_8.txt    5097_8.txt    7347_10.txt   9598_10.
txt
11848_10.txt   2848_7.txt    5098_9.txt    7348_10.txt   9599_9.t
xt
11849_8.txt    2849_7.txt    5099_7.txt    7349_7.txt    959_7.tx
t
1184_7.txt     284_10.txt    509_10.txt    734_10.txt    95_10.tx
t
11850_10.txt   2850_9.txt    50_10.txt     7350_8.txt    9600_8.t
xt
11851_10.txt   2851_10.txt   5100_10.txt   7351_10.txt   9601_10.
txt
11852_10.txt   2852_7.txt    5101_7.txt    7352_10.txt   9602_10.
txt
```

```
11853_7.txt   2853_8.txt    5102_7.txt    7353_7.txt    9603_10.
txt
11854_7.txt   2854_10.txt   5103_8.txt    7354_7.txt    9604_10.
txt
11855_7.txt   2855_8.txt    5104_10.txt   7355_10.txt   9605_10.
txt
11856_10.txt  2856_10.txt   5105_8.txt    7356_10.txt   9606_10.
txt
11857_10.txt  2857_9.txt    5106_8.txt    7357_10.txt   9607_9.t
xt
11858_9.txt   2858_10.txt   5107_10.txt   7358_10.txt   9608_10.
txt
11859_10.txt  2859_10.txt   5108_10.txt   7359_10.txt   9609_7.t
xt
1185_8.txt    285_10.txt    5109_10.txt   735_8.txt     960_7.tx
t
11860_10.txt  2860_9.txt    510_8.txt     7360_10.txt   9610_10.
txt
11861_10.txt  2861_10.txt   5110_10.txt   7361_10.txt   9611_8.t
xt
11862_10.txt  2862_10.txt   5111_10.txt   7362_10.txt   9612_8.t
xt
11863_10.txt  2863_8.txt    5112_9.txt    7363_9.txt    9613_10.
txt
11864_8.txt   2864_9.txt    5113_10.txt   7364_7.txt    9614_10.
txt
11865_10.txt  2865_10.txt   5114_10.txt   7365_7.txt    9615_9.t
xt
11866_8.txt   2866_9.txt    5115_10.txt   7366_7.txt    9616_8.t
xt
11867_8.txt   2867_10.txt   5116_10.txt   7367_7.txt    9617_10.
txt
11868_10.txt  2868_7.txt    5117_10.txt   7368_8.txt    9618_8.t
xt
11869_10.txt  2869_10.txt   5118_10.txt   7369_7.txt    9619_10.
txt
1186_8.txt    286_10.txt    5119_10.txt   736_10.txt    961_9.tx
t
11870_10.txt  2870_10.txt   511_10.txt    7370_7.txt    9620_7.t
xt
```

```
11871_7.txt    2871_10.txt    5120_10.txt    7371_7.txt     9621_8.t
xt
11872_10.txt   2872_10.txt    5121_10.txt    7372_9.txt     9622_10.
txt
11873_8.txt    2873_10.txt    5122_9.txt     7373_8.txt     9623_7.t
xt
11874_10.txt   2874_10.txt    5123_8.txt     7374_10.txt    9624_10.
txt
11875_7.txt    2875_10.txt    5124_10.txt    7375_7.txt     9625_8.t
xt
11876_10.txt   2876_10.txt    5125_7.txt     7376_8.txt     9626_8.t
xt
11877_10.txt   2877_9.txt     5126_7.txt     7377_9.txt     9627_10.
txt
11878_10.txt   2878_8.txt     5127_10.txt    7378_8.txt     9628_8.t
xt
11879_10.txt   2879_9.txt     5128_9.txt     7379_9.txt     9629_8.t
xt
1187_10.txt    287_9.txt      5129_9.txt     737_8.txt      962_8.tx
t
11880_10.txt   2880_8.txt     512_10.txt     7380_9.txt     9630_10.
txt
11881_9.txt    2881_8.txt     5130_10.txt    7381_8.txt     9631_10.
txt
11882_8.txt    2882_9.txt     5131_10.txt    7382_8.txt     9632_8.t
xt
11883_8.txt    2883_9.txt     5132_8.txt     7383_9.txt     9633_7.t
xt
11884_8.txt    2884_7.txt     5133_10.txt    7384_10.txt    9634_8.t
xt
11885_7.txt    2885_9.txt     5134_10.txt    7385_10.txt    9635_7.t
xt
11886_10.txt   2886_8.txt     5135_10.txt    7386_7.txt     9636_7.t
xt
11887_8.txt    2887_9.txt     5136_10.txt    7387_7.txt     9637_8.t
xt
11888_9.txt    2888_10.txt    5137_10.txt    7388_10.txt    9638_8.t
xt
11889_7.txt    2889_10.txt    5138_10.txt    7389_10.txt    9639_10.
txt
```

```
1188_8.txt    288_10.txt    5139_9.txt    738_8.txt     963_8.tx
t
11890_7.txt   2890_7.txt    513_10.txt    7390_10.txt   9640_9.t
xt
11891_9.txt   2891_8.txt    5140_10.txt   7391_9.txt    9641_9.t
xt
11892_9.txt   2892_10.txt   5141_9.txt    7392_8.txt    9642_10.
txt
11893_10.txt  2893_10.txt   5142_10.txt   7393_7.txt    9643_10.
txt
11894_10.txt  2894_10.txt   5143_8.txt    7394_9.txt    9644_10.
txt
11895_8.txt   2895_9.txt    5144_10.txt   7395_8.txt    9645_8.t
xt
11896_9.txt   2896_9.txt    5145_10.txt   7396_9.txt    9646_7.t
xt
11897_9.txt   2897_10.txt   5146_9.txt    7397_8.txt    9647_10.
txt
11898_8.txt   2898_9.txt    5147_8.txt    7398_10.txt   9648_8.t
xt
11899_7.txt   2899_10.txt   5148_7.txt    7399_10.txt   9649_9.t
xt
1189_9.txt    289_10.txt    5149_8.txt    739_7.txt     964_8.tx
t
118_8.txt     28_10.txt     514_10.txt    73_7.txt      9650_10.
txt
11900_8.txt   2900_9.txt    5150_7.txt    7400_9.txt    9651_9.t
xt
11901_9.txt   2901_7.txt    5151_8.txt    7401_9.txt    9652_7.t
xt
11902_9.txt   2902_9.txt    5152_9.txt    7402_8.txt    9653_9.t
xt
11903_10.txt  2903_9.txt    5153_8.txt    7403_8.txt    9654_10.
txt
11904_10.txt  2904_9.txt    5154_8.txt    7404_9.txt    9655_10.
txt
11905_8.txt   2905_7.txt    5155_9.txt    7405_10.txt   9656_10.
txt
11906_7.txt   2906_9.txt    5156_9.txt    7406_9.txt    9657_10.
txt
```

```
11907_7.txt   2907_7.txt    5157_7.txt    7407_10.txt   9658_7.t
xt
11908_8.txt   2908_8.txt    5158_10.txt   7408_8.txt    9659_8.t
xt
11909_7.txt   2909_10.txt   5159_10.txt   7409_10.txt   965_10.t
xt
1190_7.txt    290_9.txt     515_10.txt    740_7.txt     9660_9.t
xt
11910_10.txt  2910_7.txt    5160_8.txt    7410_7.txt    9661_8.t
xt
11911_10.txt  2911_10.txt   5161_8.txt    7411_10.txt   9662_10.
txt
11912_10.txt  2912_9.txt    5162_9.txt    7412_9.txt    9663_10.
txt
11913_10.txt  2913_7.txt    5163_7.txt    7413_9.txt    9664_10.
txt
11914_7.txt   2914_8.txt    5164_10.txt   7414_8.txt    9665_10.
txt
11915_10.txt  2915_7.txt    5165_9.txt    7415_8.txt    9666_10.
txt
11916_7.txt   2916_9.txt    5166_10.txt   7416_9.txt    9667_9.t
xt
11917_8.txt   2917_7.txt    5167_7.txt    7417_10.txt   9668_9.t
xt
11918_10.txt  2918_9.txt    5168_10.txt   7418_8.txt    9669_9.t
xt
11919_7.txt   2919_10.txt   5169_7.txt    7419_10.txt   966_7.tx
t
1191_9.txt    291_10.txt    516_10.txt    741_7.txt     9670_9.t
xt
11920_9.txt   2920_8.txt    5170_9.txt    7420_7.txt    9671_10.
txt
11921_10.txt  2921_10.txt   5171_10.txt   7421_7.txt    9672_7.t
xt
11922_10.txt  2922_10.txt   5172_9.txt    7422_10.txt   9673_7.t
xt
11923_10.txt  2923_10.txt   5173_9.txt    7423_9.txt    9674_9.t
xt
11924_10.txt  2924_10.txt   5174_10.txt   7424_10.txt   9675_8.t
xt
```

```
11925_7.txt    2925_10.txt    5175_10.txt    7425_9.txt     9676_10.
txt
11926_10.txt   2926_8.txt     5176_10.txt    7426_10.txt    9677_9.t
xt
11927_10.txt   2927_10.txt    5177_8.txt     7427_9.txt     9678_8.t
xt
11928_8.txt    2928_10.txt    5178_10.txt    7428_10.txt    9679_10.
txt
11929_9.txt    2929_10.txt    5179_7.txt     7429_9.txt     967_7.tx
t
1192_8.txt     292_10.txt     517_10.txt     742_9.txt      9680_8.t
xt
11930_10.txt   2930_10.txt    5180_9.txt     7430_10.txt    9681_9.t
xt
11931_9.txt    2931_10.txt    5181_10.txt    7431_10.txt    9682_10.
txt
11932_7.txt    2932_10.txt    5182_8.txt     7432_10.txt    9683_9.t
xt
11933_8.txt    2933_10.txt    5183_9.txt     7433_10.txt    9684_10.
txt
11934_8.txt    2934_10.txt    5184_9.txt     7434_7.txt     9685_10.
txt
11935_7.txt    2935_10.txt    5185_10.txt    7435_9.txt     9686_7.t
xt
11936_7.txt    2936_10.txt    5186_10.txt    7436_10.txt    9687_9.t
xt
11937_8.txt    2937_10.txt    5187_7.txt     7437_10.txt    9688_9.t
xt
11938_7.txt    2938_10.txt    5188_8.txt     7438_10.txt    9689_9.t
xt
11939_10.txt   2939_10.txt    5189_10.txt    7439_9.txt     968_10.t
xt
1193_9.txt     293_7.txt      518_10.txt     743_7.txt      9690_10.
txt
11940_7.txt    2940_10.txt    5190_8.txt     7440_7.txt     9691_10.
txt
11941_8.txt    2941_10.txt    5191_8.txt     7441_7.txt     9692_10.
txt
11942_8.txt    2942_10.txt    5192_7.txt     7442_7.txt     9693_8.t
xt
```

```
11943_7.txt    2943_10.txt    5193_9.txt     7443_7.txt     9694_9.t
xt
11944_10.txt   2944_10.txt    5194_10.txt    7444_10.txt    9695_10.
txt
11945_9.txt    2945_10.txt    5195_7.txt     7445_7.txt     9696_8.t
xt
11946_9.txt    2946_10.txt    5196_9.txt     7446_8.txt     9697_8.t
xt
11947_9.txt    2947_10.txt    5197_10.txt    7447_8.txt     9698_10.
txt
11948_10.txt   2948_10.txt    5198_7.txt     7448_8.txt     9699_10.
txt
11949_8.txt    2949_10.txt    5199_7.txt     7449_10.txt    969_7.tx
t
1194_7.txt     294_10.txt     519_10.txt     744_7.txt      96_10.tx
t
11950_8.txt    2950_10.txt    51_10.txt      7450_9.txt     9700_10.
txt
11951_8.txt    2951_8.txt     5200_10.txt    7451_9.txt     9701_10.
txt
11952_8.txt    2952_8.txt     5201_10.txt    7452_10.txt    9702_10.
txt
11953_8.txt    2953_8.txt     5202_8.txt     7453_10.txt    9703_9.t
xt
11954_9.txt    2954_10.txt    5203_7.txt     7454_10.txt    9704_10.
txt
11955_7.txt    2955_10.txt    5204_8.txt     7455_8.txt     9705_10.
txt
11956_9.txt    2956_8.txt     5205_7.txt     7456_10.txt    9706_10.
txt
11957_8.txt    2957_10.txt    5206_7.txt     7457_9.txt     9707_10.
txt
11958_9.txt    2958_10.txt    5207_7.txt     7458_10.txt    9708_10.
txt
11959_8.txt    2959_10.txt    5208_7.txt     7459_10.txt    9709_10.
txt
1195_8.txt     295_10.txt     5209_8.txt     745_10.txt     970_10.t
xt
11960_8.txt    2960_10.txt    520_8.txt      7460_10.txt    9710_10.
txt
```

```
11961_8.txt    2961_10.txt    5210_10.txt    7461_10.txt    9711_9.t
xt
11962_7.txt    2962_7.txt     5211_8.txt     7462_10.txt    9712_10.
txt
11963_10.txt   2963_8.txt     5212_7.txt     7463_10.txt    9713_10.
txt
11964_7.txt    2964_8.txt     5213_8.txt     7464_10.txt    9714_10.
txt
11965_8.txt    2965_9.txt     5214_7.txt     7465_10.txt    9715_7.t
xt
11966_10.txt   2966_10.txt    5215_10.txt    7466_10.txt    9716_10.
txt
11967_8.txt    2967_9.txt     5216_8.txt     7467_10.txt    9717_8.t
xt
11968_10.txt   2968_10.txt    5217_8.txt     7468_10.txt    9718_7.t
xt
11969_10.txt   2969_10.txt    5218_7.txt     7469_10.txt    9719_7.t
xt
1196_8.txt     296_10.txt     5219_7.txt     746_10.txt     971_8.tx
t
11970_10.txt   2970_10.txt    521_10.txt     7470_10.txt    9720_8.t
xt
11971_10.txt   2971_8.txt     5220_8.txt     7471_10.txt    9721_7.t
xt
11972_10.txt   2972_7.txt     5221_7.txt     7472_9.txt     9722_7.t
xt
11973_10.txt   2973_9.txt     5222_8.txt     7473_9.txt     9723_9.t
xt
11974_10.txt   2974_8.txt     5223_7.txt     7474_9.txt     9724_7.t
xt
11975_7.txt    2975_7.txt     5224_10.txt    7475_7.txt     9725_7.t
xt
11976_10.txt   2976_10.txt    5225_9.txt     7476_8.txt     9726_7.t
xt
11977_8.txt    2977_10.txt    5226_10.txt    7477_9.txt     9727_7.t
xt
11978_10.txt   2978_10.txt    5227_10.txt    7478_10.txt    9728_7.t
xt
11979_10.txt   2979_10.txt    5228_8.txt     7479_8.txt     9729_9.t
xt
```

```
1197_10.txt    297_10.txt    5229_10.txt    747_10.txt    972_9.tx
t
11980_10.txt   2980_7.txt    522_8.txt      7480_10.txt   9730_8.t
xt
11981_9.txt    2981_10.txt   5230_10.txt    7481_10.txt   9731_8.t
xt
11982_7.txt    2982_8.txt    5231_10.txt    7482_10.txt   9732_10.
txt
11983_10.txt   2983_7.txt    5232_8.txt     7483_10.txt   9733_8.t
xt
11984_10.txt   2984_8.txt    5233_10.txt    7484_10.txt   9734_7.t
xt
11985_10.txt   2985_8.txt    5234_10.txt    7485_8.txt    9735_8.t
xt
11986_8.txt    2986_8.txt    5235_8.txt     7486_8.txt    9736_8.t
xt
11987_7.txt    2987_10.txt   5236_10.txt    7487_8.txt    9737_9.t
xt
11988_8.txt    2988_7.txt    5237_10.txt    7488_7.txt    9738_8.t
xt
11989_10.txt   2989_10.txt   5238_7.txt     7489_7.txt    9739_7.t
xt
1198_10.txt    298_8.txt     5239_7.txt     748_9.txt     973_9.tx
t
11990_10.txt   2990_10.txt   523_10.txt     7490_8.txt    9740_7.t
xt
11991_10.txt   2991_7.txt    5240_7.txt     7491_7.txt    9741_8.t
xt
11992_10.txt   2992_7.txt    5241_7.txt     7492_7.txt    9742_10.
txt
11993_9.txt    2993_10.txt   5242_7.txt     7493_7.txt    9743_8.t
xt
11994_10.txt   2994_8.txt    5243_9.txt     7494_10.txt   9744_7.t
xt
11995_10.txt   2995_7.txt    5244_7.txt     7495_7.txt    9745_7.t
xt
11996_10.txt   2996_7.txt    5245_8.txt     7496_8.txt    9746_9.t
xt
11997_10.txt   2997_7.txt    5246_10.txt    7497_10.txt   9747_10.
txt
```

```
11998_9.txt   2998_7.txt   5247_10.txt   7498_10.txt   9748_10.
txt
11999_10.txt  2999_7.txt   5248_8.txt    7499_7.txt    9749_9.t
xt
1199_10.txt   299_10.txt   5249_9.txt    749_10.txt    974_10.t
xt
119_10.txt    29_10.txt    524_10.txt    74_8.txt      9750_10.
txt
11_9.txt      2_9.txt      5250_10.txt   7500_8.txt    9751_7.t
xt
12000_8.txt   3000_8.txt   5251_7.txt    7501_8.txt    9752_10.
txt
12001_8.txt   3001_10.txt  5252_9.txt    7502_7.txt    9753_10.
txt
12002_10.txt  3002_8.txt   5253_10.txt   7503_7.txt    9754_10.
txt
12003_10.txt  3003_9.txt   5254_9.txt    7504_7.txt    9755_7.t
xt
12004_10.txt  3004_10.txt  5255_10.txt   7505_9.txt    9756_10.
txt
12005_10.txt  3005_8.txt   5256_10.txt   7506_7.txt    9757_8.t
xt
12006_10.txt  3006_9.txt   5257_10.txt   7507_8.txt    9758_10.
txt
12007_10.txt  3007_7.txt   5258_10.txt   7508_7.txt    9759_10.
txt
12008_8.txt   3008_7.txt   5259_9.txt    7509_8.txt    975_9.tx
t
12009_10.txt  3009_8.txt   525_10.txt    750_8.txt     9760_8.t
xt
1200_10.txt   300_9.txt    5260_10.txt   7510_8.txt    9761_10.
txt
12010_10.txt  3010_8.txt   5261_10.txt   7511_9.txt    9762_8.t
xt
12011_10.txt  3011_7.txt   5262_7.txt    7512_8.txt    9763_8.t
xt
12012_10.txt  3012_8.txt   5263_8.txt    7513_9.txt    9764_9.t
xt
12013_10.txt  3013_8.txt   5264_7.txt    7514_8.txt    9765_10.
txt
```

```
12014_10.txt  3014_8.txt   5265_9.txt   7515_8.txt   9766_9.t
xt
12015_10.txt  3015_9.txt   5266_10.txt  7516_7.txt   9767_7.t
xt
12016_7.txt   3016_10.txt  5267_7.txt   7517_7.txt   9768_10.
txt
12017_7.txt   3017_7.txt   5268_10.txt  7518_9.txt   9769_8.t
xt
12018_7.txt   3018_10.txt  5269_10.txt  7519_9.txt   976_8.tx
t
12019_8.txt   3019_8.txt   526_10.txt   751_9.txt    9770_10.
txt
1201_8.txt    301_10.txt   5270_7.txt   7520_8.txt   9771_10.
txt
12020_7.txt   3020_9.txt   5271_7.txt   7521_7.txt   9772_10.
txt
12021_9.txt   3021_8.txt   5272_7.txt   7522_8.txt   9773_7.t
xt
12022_10.txt  3022_7.txt   5273_7.txt   7523_7.txt   9774_8.t
xt
12023_9.txt   3023_7.txt   5274_7.txt   7524_9.txt   9775_8.t
xt
12024_7.txt   3024_9.txt   5275_10.txt  7525_7.txt   9776_8.t
xt
12025_7.txt   3025_7.txt   5276_10.txt  7526_8.txt   9777_8.t
xt
12026_8.txt   3026_7.txt   5277_10.txt  7527_10.txt  9778_8.t
xt
12027_10.txt  3027_8.txt   5278_7.txt   7528_10.txt  9779_9.t
xt
12028_10.txt  3028_7.txt   5279_8.txt   7529_10.txt  977_8.tx
t
12029_10.txt  3029_8.txt   527_9.txt    752_7.txt    9780_10.
txt
1202_9.txt    302_10.txt   5280_8.txt   7530_9.txt   9781_8.t
xt
12030_9.txt   3030_9.txt   5281_10.txt  7531_8.txt   9782_10.
txt
12031_7.txt   3031_8.txt   5282_10.txt  7532_7.txt   9783_9.t
xt
```

```
12032_8.txt    3032_7.txt    5283_7.txt    7533_10.txt    9784_8.t
xt
12033_8.txt    3033_8.txt    5284_9.txt    7534_10.txt    9785_7.t
xt
12034_10.txt   3034_7.txt    5285_7.txt    7535_10.txt    9786_8.t
xt
12035_8.txt    3035_9.txt    5286_10.txt   7536_10.txt    9787_7.t
xt
12036_7.txt    3036_9.txt    5287_8.txt    7537_10.txt    9788_9.t
xt
12037_10.txt   3037_7.txt    5288_7.txt    7538_7.txt     9789_10.
txt
12038_9.txt    3038_8.txt    5289_10.txt   7539_10.txt    978_9.tx
t
12039_10.txt   3039_8.txt    528_9.txt     753_8.txt      9790_9.t
xt
1203_8.txt     303_10.txt    5290_10.txt   7540_9.txt     9791_9.t
xt
12040_7.txt    3040_8.txt    5291_8.txt    7541_10.txt    9792_8.t
xt
12041_9.txt    3041_7.txt    5292_7.txt    7542_10.txt    9793_7.t
xt
12042_10.txt   3042_8.txt    5293_8.txt    7543_8.txt     9794_7.t
xt
12043_10.txt   3043_8.txt    5294_8.txt    7544_7.txt     9795_7.t
xt
12044_7.txt    3044_7.txt    5295_8.txt    7545_8.txt     9796_9.t
xt
12045_10.txt   3045_10.txt   5296_10.txt   7546_7.txt     9797_7.t
xt
12046_10.txt   3046_10.txt   5297_8.txt    7547_10.txt    9798_7.t
xt
12047_10.txt   3047_7.txt    5298_8.txt    7548_8.txt     9799_7.t
xt
12048_10.txt   3048_10.txt   5299_8.txt    7549_7.txt     979_8.tx
t
12049_10.txt   3049_9.txt    529_10.txt    754_9.txt      97_9.txt
1204_10.txt    304_10.txt    52_10.txt     7550_9.txt     9800_9.t
xt
12050_10.txt   3050_9.txt    5300_8.txt    7551_10.txt    9801_10.
```

```
txt
12051_10.txt  3051_9.txt   5301_8.txt   7552_10.txt  9802_10.
txt
12052_9.txt   3052_10.txt  5302_9.txt   7553_10.txt  9803_7.t
xt
12053_9.txt   3053_8.txt   5303_10.txt  7554_10.txt  9804_10.
txt
12054_10.txt  3054_7.txt   5304_10.txt  7555_7.txt   9805_10.
txt
12055_10.txt  3055_9.txt   5305_7.txt   7556_8.txt   9806_8.t
xt
12056_10.txt  3056_8.txt   5306_7.txt   7557_9.txt   9807_10.
txt
12057_10.txt  3057_8.txt   5307_7.txt   7558_8.txt   9808_9.t
xt
12058_9.txt   3058_9.txt   5308_10.txt  7559_10.txt  9809_7.t
xt
12059_9.txt   3059_8.txt   5309_7.txt   755_10.txt   980_7.tx
t
1205_8.txt    305_8.txt    530_10.txt   7560_9.txt   9810_7.t
xt
12060_9.txt   3060_10.txt  5310_10.txt  7561_9.txt   9811_7.t
xt
12061_10.txt  3061_10.txt  5311_7.txt   7562_10.txt  9812_7.t
xt
12062_10.txt  3062_10.txt  5312_7.txt   7563_10.txt  9813_8.t
xt
12063_10.txt  3063_10.txt  5313_7.txt   7564_10.txt  9814_10.
txt
12064_10.txt  3064_8.txt   5314_10.txt  7565_10.txt  9815_7.t
xt
12065_7.txt   3065_10.txt  5315_9.txt   7566_10.txt  9816_10.
txt
12066_8.txt   3066_10.txt  5316_7.txt   7567_10.txt  9817_8.t
xt
12067_9.txt   3067_9.txt   5317_7.txt   7568_10.txt  9818_7.t
xt
12068_9.txt   3068_9.txt   5318_7.txt   7569_10.txt  9819_10.
txt
12069_10.txt  3069_9.txt   5319_8.txt   756_10.txt   981_7.tx
```

```
t
1206_10.txt   306_10.txt    531_10.txt    7570_10.txt   9820_10.
txt
12070_7.txt   3070_7.txt    5320_10.txt   7571_8.txt    9821_8.t
xt
12071_8.txt   3071_9.txt    5321_10.txt   7572_10.txt   9822_10.
txt
12072_10.txt  3072_8.txt    5322_10.txt   7573_9.txt    9823_7.t
xt
12073_9.txt   3073_8.txt    5323_7.txt    7574_10.txt   9824_10.
txt
12074_10.txt  3074_8.txt    5324_10.txt   7575_10.txt   9825_10.
txt
12075_7.txt   3075_9.txt    5325_7.txt    7576_10.txt   9826_10.
txt
12076_8.txt   3076_8.txt    5326_10.txt   7577_8.txt    9827_10.
txt
12077_8.txt   3077_10.txt   5327_10.txt   7578_8.txt    9828_10.
txt
12078_8.txt   3078_10.txt   5328_8.txt    7579_8.txt    9829_7.t
xt
12079_7.txt   3079_10.txt   5329_9.txt    757_8.txt     982_8.tx
t
1207_10.txt   307_8.txt     532_9.txt     7580_7.txt    9830_7.t
xt
12080_8.txt   3080_10.txt   5330_7.txt    7581_8.txt    9831_8.t
xt
12081_8.txt   3081_9.txt    5331_10.txt   7582_8.txt    9832_10.
txt
12082_8.txt   3082_10.txt   5332_7.txt    7583_7.txt    9833_8.t
xt
12083_8.txt   3083_10.txt   5333_10.txt   7584_7.txt    9834_10.
txt
12084_8.txt   3084_10.txt   5334_7.txt    7585_8.txt    9835_9.t
xt
12085_8.txt   3085_10.txt   5335_10.txt   7586_10.txt   9836_9.t
xt
12086_10.txt  3086_10.txt   5336_8.txt    7587_9.txt    9837_7.t
xt
12087_10.txt  3087_10.txt   5337_9.txt    7588_8.txt    9838_7.t
```

```
xt
12088_9.txt    3088_8.txt     5338_10.txt    7589_8.txt     9839_7.t
xt
12089_10.txt   3089_9.txt     5339_8.txt     758_9.txt      983_7.tx
t
1208_9.txt     308_8.txt      533_10.txt     7590_10.txt    9840_10.
txt
12090_9.txt    3090_10.txt    5340_8.txt     7591_8.txt     9841_7.t
xt
12091_8.txt    3091_10.txt    5341_10.txt    7592_9.txt     9842_7.t
xt
12092_7.txt    3092_10.txt    5342_10.txt    7593_7.txt     9843_7.t
xt
12093_8.txt    3093_10.txt    5343_8.txt     7594_8.txt     9844_8.t
xt
12094_7.txt    3094_10.txt    5344_7.txt     7595_10.txt    9845_8.t
xt
12095_8.txt    3095_10.txt    5345_9.txt     7596_9.txt     9846_7.t
xt
12096_8.txt    3096_10.txt    5346_7.txt     7597_9.txt     9847_7.t
xt
12097_7.txt    3097_9.txt     5347_7.txt     7598_10.txt    9848_7.t
xt
12098_8.txt    3098_10.txt    5348_8.txt     7599_10.txt    9849_9.t
xt
12099_9.txt    3099_10.txt    5349_7.txt     759_10.txt     984_7.tx
t
1209_10.txt    309_9.txt      534_10.txt     75_8.txt       9850_7.t
xt
120_8.txt      30_7.txt       5350_9.txt     7600_10.txt    9851_7.t
xt
12100_8.txt    3100_10.txt    5351_7.txt     7601_10.txt    9852_7.t
xt
12101_7.txt    3101_9.txt     5352_7.txt     7602_10.txt    9853_8.t
xt
12102_8.txt    3102_9.txt     5353_7.txt     7603_10.txt    9854_7.t
xt
12103_7.txt    3103_7.txt     5354_10.txt    7604_9.txt     9855_7.t
xt
12104_7.txt    3104_10.txt    5355_8.txt     7605_8.txt     9856_8.t
```

```
xt
12105_10.txt  3105_8.txt    5356_8.txt    7606_9.txt    9857_7.t
xt
12106_10.txt  3106_8.txt    5357_8.txt    7607_8.txt    9858_7.t
xt
12107_10.txt  3107_8.txt    5358_10.txt   7608_10.txt   9859_10.
txt
12108_9.txt   3108_8.txt    5359_10.txt   7609_10.txt   985_7.tx
t
12109_10.txt  3109_10.txt   535_10.txt    760_7.txt     9860_7.t
xt
1210_8.txt    310_7.txt     5360_10.txt   7610_9.txt    9861_10.
txt
12110_8.txt   3110_8.txt    5361_10.txt   7611_9.txt    9862_9.t
xt
12111_7.txt   3111_7.txt    5362_8.txt    7612_9.txt    9863_10.
txt
12112_10.txt  3112_7.txt    5363_9.txt    7613_9.txt    9864_8.t
xt
12113_8.txt   3113_9.txt    5364_10.txt   7614_10.txt   9865_8.t
xt
12114_7.txt   3114_9.txt    5365_10.txt   7615_10.txt   9866_7.t
xt
12115_10.txt  3115_8.txt    5366_10.txt   7616_10.txt   9867_10.
txt
12116_9.txt   3116_10.txt   5367_9.txt    7617_9.txt    9868_9.t
xt
12117_7.txt   3117_8.txt    5368_9.txt    7618_10.txt   9869_10.
txt
12118_7.txt   3118_9.txt    5369_9.txt    7619_10.txt   986_10.t
xt
12119_7.txt   3119_8.txt    536_10.txt    761_10.txt    9870_9.t
xt
1211_7.txt    311_9.txt     5370_7.txt    7620_10.txt   9871_7.t
xt
12120_10.txt  3120_8.txt    5371_8.txt    7621_10.txt   9872_10.
txt
12121_7.txt   3121_10.txt   5372_7.txt    7622_10.txt   9873_7.t
xt
12122_7.txt   3122_8.txt    5373_7.txt    7623_9.txt    9874_8.t
```

```
xt
12123_8.txt   3123_10.txt   5374_8.txt   7624_7.txt   9875_7.t
xt
12124_8.txt   3124_9.txt    5375_8.txt   7625_10.txt  9876_8.t
xt
12125_8.txt   3125_10.txt   5376_7.txt   7626_9.txt   9877_8.t
xt
12126_8.txt   3126_10.txt   5377_7.txt   7627_10.txt  9878_7.t
xt
12127_8.txt   3127_9.txt    5378_7.txt   7628_8.txt   9879_8.t
xt
12128_8.txt   3128_9.txt    5379_7.txt   7629_10.txt  987_8.tx
t
12129_7.txt   3129_10.txt   537_10.txt   762_9.txt    9880_10.
txt
1212_8.txt    312_10.txt    5380_7.txt   7630_8.txt   9881_8.t
xt
12130_7.txt   3130_9.txt    5381_7.txt   7631_10.txt  9882_8.t
xt
12131_7.txt   3131_7.txt    5382_7.txt   7632_10.txt  9883_8.t
xt
12132_10.txt  3132_9.txt    5383_10.txt  7633_8.txt   9884_10.
txt
12133_7.txt   3133_9.txt    5384_8.txt   7634_7.txt   9885_10.
txt
12134_8.txt   3134_8.txt    5385_7.txt   7635_10.txt  9886_10.
txt
12135_10.txt  3135_9.txt    5386_7.txt   7636_8.txt   9887_9.t
xt
12136_9.txt   3136_8.txt    5387_8.txt   7637_7.txt   9888_10.
txt
12137_10.txt  3137_8.txt    5388_8.txt   7638_7.txt   9889_9.t
xt
12138_7.txt   3138_9.txt    5389_7.txt   7639_8.txt   988_8.tx
t
12139_10.txt  3139_10.txt   538_10.txt   763_10.txt   9890_8.t
xt
1213_10.txt   313_10.txt    5390_9.txt   7640_10.txt  9891_10.
txt
12140_8.txt   3140_7.txt    5391_9.txt   7641_10.txt  9892_8.t
```

```
xt
12141_8.txt   3141_10.txt   5392_10.txt   7642_9.txt    9893_7.t
xt
12142_9.txt   3142_8.txt    5393_10.txt   7643_7.txt    9894_8.t
xt
12143_8.txt   3143_10.txt   5394_10.txt   7644_7.txt    9895_8.t
xt
12144_8.txt   3144_10.txt   5395_10.txt   7645_7.txt    9896_8.t
xt
12145_10.txt  3145_10.txt   5396_9.txt    7646_8.txt    9897_10.
txt
12146_7.txt   3146_10.txt   5397_9.txt    7647_7.txt    9898_10.
txt
12147_10.txt  3147_10.txt   5398_8.txt    7648_7.txt    9899_7.t
xt
12148_10.txt  3148_9.txt    5399_8.txt    7649_9.txt    989_9.tx
t
12149_10.txt  3149_8.txt    539_10.txt    764_10.txt    98_10.tx
t
1214_10.txt   314_10.txt    53_10.txt     7650_9.txt    9900_10.
txt
12150_7.txt   3150_10.txt   5400_10.txt   7651_8.txt    9901_8.t
xt
12151_10.txt  3151_9.txt    5401_9.txt    7652_8.txt    9902_7.t
xt
12152_9.txt   3152_9.txt    5402_9.txt    7653_10.txt   9903_9.t
xt
12153_10.txt  3153_10.txt   5403_10.txt   7654_10.txt   9904_8.t
xt
12154_7.txt   3154_10.txt   5404_10.txt   7655_10.txt   9905_10.
txt
12155_7.txt   3155_10.txt   5405_10.txt   7656_10.txt   9906_10.
txt
12156_8.txt   3156_9.txt    5406_7.txt    7657_10.txt   9907_7.t
xt
12157_9.txt   3157_10.txt   5407_7.txt    7658_10.txt   9908_8.t
xt
12158_7.txt   3158_8.txt    5408_7.txt    7659_10.txt   9909_7.t
xt
12159_7.txt   3159_9.txt    5409_10.txt   765_9.txt     990_9.tx
```

```
t
1215_10.txt    315_10.txt     540_8.txt      7660_10.txt    9910_10.
txt
12160_10.txt   3160_7.txt     5410_7.txt     7661_10.txt    9911_10.
txt
12161_7.txt    3161_10.txt    5411_10.txt    7662_10.txt    9912_10.
txt
12162_8.txt    3162_8.txt     5412_9.txt     7663_7.txt     9913_10.
txt
12163_8.txt    3163_10.txt    5413_10.txt    7664_10.txt    9914_7.t
xt
12164_9.txt    3164_10.txt    5414_10.txt    7665_7.txt     9915_8.t
xt
12165_10.txt   3165_10.txt    5415_10.txt    7666_7.txt     9916_7.t
xt
12166_10.txt   3166_9.txt     5416_9.txt     7667_7.txt     9917_10.
txt
12167_10.txt   3167_7.txt     5417_10.txt    7668_10.txt    9918_10.
txt
12168_10.txt   3168_7.txt     5418_10.txt    7669_8.txt     9919_9.t
xt
12169_7.txt    3169_8.txt     5419_10.txt    766_10.txt     991_7.tx
t
1216_10.txt    316_10.txt     541_7.txt      7670_8.txt     9920_7.t
xt
12170_8.txt    3170_9.txt     5420_7.txt     7671_10.txt    9921_8.t
xt
12171_7.txt    3171_9.txt     5421_7.txt     7672_10.txt    9922_10.
txt
12172_7.txt    3172_9.txt     5422_9.txt     7673_7.txt     9923_7.t
xt
12173_8.txt    3173_8.txt     5423_9.txt     7674_10.txt    9924_9.t
xt
12174_7.txt    3174_9.txt     5424_10.txt    7675_8.txt     9925_7.t
xt
12175_7.txt    3175_8.txt     5425_10.txt    7676_9.txt     9926_7.t
xt
12176_8.txt    3176_9.txt     5426_10.txt    7677_10.txt    9927_9.t
xt
12177_7.txt    3177_9.txt     5427_10.txt    7678_10.txt    9928_7.t
```

```
xt
12178_7.txt    3178_8.txt     5428_10.txt    7679_10.txt    9929_7.t
xt
12179_8.txt    3179_8.txt     5429_10.txt    767_9.txt      992_7.tx
t
1217_10.txt    317_10.txt     542_9.txt      7680_8.txt     9930_8.t
xt
12180_10.txt   3180_8.txt     5430_10.txt    7681_9.txt     9931_9.t
xt
12181_8.txt    3181_10.txt    5431_10.txt    7682_8.txt     9932_8.t
xt
12182_8.txt    3182_8.txt     5432_7.txt     7683_10.txt    9933_8.t
xt
12183_10.txt   3183_9.txt     5433_10.txt    7684_7.txt     9934_8.t
xt
12184_10.txt   3184_8.txt     5434_9.txt     7685_10.txt    9935_7.t
xt
12185_9.txt    3185_10.txt    5435_10.txt    7686_10.txt    9936_10.
txt
12186_9.txt    3186_8.txt     5436_7.txt     7687_8.txt     9937_10.
txt
12187_9.txt    3187_7.txt     5437_10.txt    7688_10.txt    9938_9.t
xt
12188_10.txt   3188_8.txt     5438_10.txt    7689_10.txt    9939_10.
txt
12189_7.txt    3189_8.txt     5439_10.txt    768_9.txt      993_8.tx
t
1218_8.txt     318_10.txt     543_10.txt     7690_9.txt     9940_9.t
xt
12190_7.txt    3190_7.txt     5440_10.txt    7691_9.txt     9941_7.t
xt
12191_7.txt    3191_8.txt     5441_10.txt    7692_7.txt     9942_7.t
xt
12192_7.txt    3192_10.txt    5442_8.txt     7693_9.txt     9943_7.t
xt
12193_10.txt   3193_9.txt     5443_8.txt     7694_10.txt    9944_9.t
xt
12194_10.txt   3194_9.txt     5444_8.txt     7695_9.txt     9945_8.t
xt
12195_8.txt    3195_10.txt    5445_10.txt    7696_8.txt     9946_9.t
```

```
xt
12196_10.txt  3196_10.txt   5446_9.txt    7697_9.txt    9947_10.
txt
12197_9.txt   3197_10.txt   5447_10.txt   7698_10.txt   9948_8.t
xt
12198_10.txt  3198_10.txt   5448_10.txt   7699_10.txt   9949_9.t
xt
12199_9.txt   3199_10.txt   5449_8.txt    769_8.txt     994_7.tx
t
1219_10.txt   319_9.txt     544_8.txt     76_7.txt      9950_8.t
xt
121_10.txt    31_8.txt      5450_10.txt   7700_10.txt   9951_7.t
xt
12200_8.txt   3200_10.txt   5451_8.txt    7701_10.txt   9952_8.t
xt
12201_8.txt   3201_10.txt   5452_8.txt    7702_10.txt   9953_10.
txt
12202_9.txt   3202_10.txt   5453_8.txt    7703_8.txt    9954_8.t
xt
12203_10.txt  3203_10.txt   5454_7.txt    7704_10.txt   9955_9.t
xt
12204_8.txt   3204_10.txt   5455_7.txt    7705_7.txt    9956_9.t
xt
12205_8.txt   3205_8.txt    5456_10.txt   7706_7.txt    9957_7.t
xt
12206_9.txt   3206_8.txt    5457_8.txt    7707_9.txt    9958_7.t
xt
12207_8.txt   3207_7.txt    5458_10.txt   7708_10.txt   9959_7.t
xt
12208_8.txt   3208_7.txt    5459_8.txt    7709_7.txt    995_9.tx
t
12209_7.txt   3209_8.txt    545_10.txt    770_10.txt    9960_7.t
xt
1220_9.txt    320_8.txt     5460_8.txt    7710_8.txt    9961_9.t
xt
12210_9.txt   3210_8.txt    5461_7.txt    7711_10.txt   9962_9.t
xt
12211_10.txt  3211_7.txt    5462_8.txt    7712_10.txt   9963_10.
txt
12212_10.txt  3212_7.txt    5463_9.txt    7713_10.txt   9964_10.
```

```
txt
12213_10.txt  3213_10.txt  5464_7.txt   7714_10.txt  9965_10.
txt
12214_10.txt  3214_10.txt  5465_8.txt   7715_8.txt   9966_10.
txt
12215_10.txt  3215_10.txt  5466_8.txt   7716_9.txt   9967_10.
txt
12216_8.txt   3216_8.txt   5467_9.txt   7717_10.txt  9968_9.t
xt
12217_10.txt  3217_8.txt   5468_7.txt   7718_10.txt  9969_10.
txt
12218_7.txt   3218_9.txt   5469_7.txt   7719_9.txt   996_9.tx
t
12219_10.txt  3219_10.txt  546_10.txt   771_7.txt    9970_10.
txt
1221_7.txt    321_10.txt   5470_10.txt  7720_8.txt   9971_10.
txt
12220_8.txt   3220_10.txt  5471_8.txt   7721_7.txt   9972_10.
txt
12221_8.txt   3221_10.txt  5472_7.txt   7722_10.txt  9973_8.t
xt
12222_10.txt  3222_7.txt   5473_8.txt   7723_10.txt  9974_8.t
xt
12223_10.txt  3223_8.txt   5474_9.txt   7724_10.txt  9975_10.
txt
12224_9.txt   3224_7.txt   5475_8.txt   7725_10.txt  9976_7.t
xt
12225_7.txt   3225_10.txt  5476_8.txt   7726_10.txt  9977_7.t
xt
12226_8.txt   3226_10.txt  5477_7.txt   7727_9.txt   9978_8.t
xt
12227_7.txt   3227_7.txt   5478_9.txt   7728_7.txt   9979_7.t
xt
12228_8.txt   3228_7.txt   5479_10.txt  7729_7.txt   997_7.tx
t
12229_9.txt   3229_7.txt   547_10.txt   772_10.txt   9980_8.t
xt
1222_10.txt   322_10.txt   5480_10.txt  7730_7.txt   9981_7.t
xt
12230_9.txt   3230_8.txt   5481_10.txt  7731_9.txt   9982_9.t
```

```
xt
12231_10.txt  3231_10.txt   5482_10.txt   7732_8.txt    9983_7.t
xt
12232_8.txt   3232_9.txt    5483_10.txt   7733_9.txt    9984_9.t
xt
12233_8.txt   3233_10.txt   5484_10.txt   7734_10.txt   9985_9.t
xt
12234_8.txt   3234_9.txt    5485_10.txt   7735_10.txt   9986_9.t
xt
12235_7.txt   3235_9.txt    5486_10.txt   7736_10.txt   9987_9.t
xt
12236_10.txt  3236_7.txt    5487_10.txt   7737_10.txt   9988_8.t
xt
12237_10.txt  3237_8.txt    5488_10.txt   7738_8.txt    9989_9.t
xt
12238_8.txt   3238_9.txt    5489_10.txt   7739_10.txt   998_7.tx
t
12239_10.txt  3239_10.txt   548_7.txt     773_7.txt     9990_8.t
xt
1223_7.txt    323_10.txt    5490_9.txt    7740_10.txt   9991_10.
txt
12240_8.txt   3240_10.txt   5491_10.txt   7741_9.txt    9992_10.
txt
12241_10.txt  3241_8.txt    5492_10.txt   7742_8.txt    9993_10.
txt
12242_10.txt  3242_8.txt    5493_10.txt   7743_8.txt    9994_10.
txt
12243_8.txt   3243_8.txt    5494_10.txt   7744_10.txt   9995_10.
txt
12244_7.txt   3244_10.txt   5495_8.txt    7745_8.txt    9996_9.t
xt
12245_10.txt  3245_10.txt   5496_9.txt    7746_10.txt   9997_7.t
xt
12246_9.txt   3246_9.txt    5497_9.txt    7747_10.txt   9998_9.t
xt
12247_8.txt   3247_10.txt   5498_7.txt    7748_9.txt    9999_8.t
xt
12248_7.txt   3248_10.txt   5499_10.txt   7749_8.txt    999_10.t
xt
12249_8.txt   3249_9.txt    549_9.txt     774_8.txt     99_8.txt
```

```
1224_9.txt    324_8.txt     54_10.txt     7750_8.txt    9_7.txt
```

```
cat 0_9.txt
```

```
Bromwell High is a cartoon comedy. It ran at the same time as so
me other programs about school life, such as "Teachers". My 35 y
ears in the teaching profession lead me to believe that Bromwell
 High's satire is much closer to reality than is "Teachers". The
 scramble to survive financially, the insightful students who ca
n see right through their pathetic teachers' pomp, the pettiness
 of the whole situation, all remind me of the schools I knew and
 their students. When I saw the episode in which a student repea
tedly tried to burn down the school, I immediately recalled ....
..... at .......... High. A classic line: INSPECTOR: I'm here to
 sack one of your teachers. STUDENT: Welcome to Bromwell High. I
 expect that many adults of my age think that Bromwell High is f
ar fetched. What a pity that it isn't!
```

```
# pip install pyprind
import pyprind   # module for Python Progress Indicator
import pandas as pd
import os

labels = {'pos':1, 'neg':0}  # 1 = positive and 0 = negative
pbar = pyprind.ProgBar(50000)
df = pd.DataFrame()
for s in ('test', 'train'):
    for l in ('pos', 'neg'):
        path ='data/aclImdb/%s/%s' % (s, l)
        for file in os.listdir(path):
            with open(os.path.join(path, file), 'r') as infile:
                txt = infile.read()
            df = df.append([[txt, labels[l]]], ignore_index=True
)
            pbar.update()
df.columns = ['review', 'sentiment']
```

```
0%                          100%
[##############################] | ETA: 00:00:00
Total time elapsed: 00:01:41
```

```
df.head(3)
```

	review	sentiment
0	I went and saw this movie last night after bei...	1
1	Actor turned director Bill Paxton follows up h...	1
2	As a recreational golfer with some knowledge o...	1

Shuffling the DataFrame:

```
import numpy as np
np.random.seed(0)
df = df.reindex(np.random.permutation(df.index))
```

```
df.head(3)
```

	review	sentiment
11841	In 1974, the teenager Martha Moxley (Maggie Gr...	1
19602	OK... so... I really like Kris Kristofferson a...	0
45519	***SPOILER*** Do not read this, if you think a...	0

Optional: Saving the assembled data as CSV file:

```
df.to_csv('./data/movie_data.csv', index=False, encoding='utf-8'
)
```

```
import pandas as pd
df = pd.read_csv('./data/movie_data.csv', encoding='utf-8')
df.head(3)
```

	review	sentiment
0	In 1974, the teenager Martha Moxley (Maggie Gr...	1
1	OK... so... I really like Kris Kristofferson a...	0
2	***SPOILER*** Do not read this, if you think a...	0

Text feature extraction

[back to top]

Bag-of-words model

[back to top]

Free text with variables length is very far from the fixed length numeric representation that we need to do machine learning with scikit-learn. However, there is an easy and effective way to go from text data to a numeric representation using the so-called bag-of-words model, which provides a data structure that is compatible with the machine learning aglorithms in scikit-learn.

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
count = CountVectorizer()
docs = np.array([
        'The sun is shining',
        'The weather is sweet',
        'The sun is shining, the weather is sweet, and one and o
ne is two'])
bag = count.fit_transform(docs)
```

```
count.vocabulary_
```

```
{u'and': 0,
 u'is': 1,
 u'one': 2,
 u'shining': 3,
 u'sun': 4,
 u'sweet': 5,
 u'the': 6,
 u'two': 7,
 u'weather': 8}
```

```
count.get_feature_names()
```

```
[u'and',
 u'is',
 u'one',
 u'shining',
 u'sun',
 u'sweet',
 u'the',
 u'two',
 u'weather']
```

As we can see from executing the preceding command, the vocabulary is stored in a Python dictionary, which maps the unique words that are mapped to integer indices. Next let us print the feature vectors that we just created:

```
bag.toarray()  # 每行对应着一个 document， 每列对应一个 word， 值是 word 对应的计数
```

```
array([[0, 1, 0, 1, 1, 0, 1, 0, 0],
       [0, 1, 0, 0, 0, 1, 1, 0, 1],
       [2, 3, 2, 1, 1, 1, 2, 1, 1]])
```

```
count.inverse_transform(bag)
```

```
[array([u'the', u'sun', u'is', u'shining'],
       dtype='<U7'), array([u'the', u'is', u'weather', u'sweet']
,
       dtype='<U7'), array([u'the', u'sun', u'is', u'shining', u
'weather', u'sweet', u'and',
        u'one', u'two'],
       dtype='<U7')]
```

Bigrams and N-Grams

[back to top]

In last section, we used the so-called 1-gram (unigram) tokenization: Each token represents a single element with regard to the splittling criterion.

Entirely discarding word order is not always a good idea, as composite phrases often have specific meaning, and modifiers like "not" can invert the meaning of words.

A simple way to include some word order are n-grams, which don't only look at a single token, but at all pairs of neighborhing tokens. For example, in 2-gram (bigram) tokenization, we would group words together with an overlap of one word; in 3-gram (trigram) splits we would create an overlap two words, and so forth:

- original text: "this is how you get ants"
- 1-gram: "this", "is", "how", "you", "get", "ants"
- 2-gram: "this is", "is how", "how you", "you get", "get ants"
- 3-gram: "this is how", "is how you", "how you get", "you get ants"

Which "n" we choose for "n-gram" tokenization to obtain the optimal performance in our predictive model depends on the learning algorithm, dataset, and task. Or in other words, we have consider "n" in "n-grams" as a tuning parameters.

The CountVectorizer class in scikit-learn allows us to use different n-gram models via its ngram_range parameter. While a 1-gram representation is used by default, we could switch to a 2-gram representation by initializing a new CountVectorizer instance with ngram_range=(2,2)

```
bigram_vectorizer = CountVectorizer(ngram_range=(2,2))
bigram_vectorizer.fit_transform(docs).toarray()
```

```
array([[0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

```
bigram_vectorizer.vocabulary_
```

```
{u'and one': 0,
 u'is shining': 1,
 u'is sweet': 2,
 u'is two': 3,
 u'one and': 4,
 u'one is': 5,
 u'shining the': 6,
 u'sun is': 7,
 u'sweet and': 8,
 u'the sun': 9,
 u'the weather': 10,
 u'weather is': 11}
```

Character n-grams

[back to top]

Sometimes it is also helpful not only to look at words, but to consider single characters instead.
That is particularly useful if we have very noisy data and want to identify the language, or if we want to predict something about a single word. We can simply look at characters instead of words by setting `analyzer="char"` .

```
X = ['Some say the world will end in fire,', 'Some say in ice.']
```

```
char_vectorizer = CountVectorizer(analyzer="char")
char_vectorizer.fit(X)
```

```
CountVectorizer(analyzer='char', binary=False, decode_error=u'st
rict',
        dtype=<type 'numpy.int64'>, encoding=u'utf-8', input=u'c
ontent',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern=u'(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

```
print(char_vectorizer.get_feature_names())
```

```
[u' ', u',', u'.', u'a', u'c', u'd', u'e', u'f', u'h', u'i', u'l
', u'm', u'n', u'o', u'r', u's', u't', u'w', u'y']
```

Tfidf encoding

[back to top]

```
np.set_printoptions(precision=2)
```

When we are analyzing text data, we often encounter words that occur across multiple documents from both classes. Those frequently occurring words typically don't contain useful or discriminatory information. In this subsection, we will learn about a useful technique called term frequency-inverse document frequency (tf-idf) that can be used to downweight those frequently occurring words in the feature vectors. The tf-idf can be de ned as the product of the term frequency and the inverse document frequency:

$$\text{tf-idf}(t, d) = \text{tf (t,d)} \times \text{idf}(t, d)$$

Here the tf(t, d) is the term frequency that we introduced in the previous section, and the inverse document frequency *idf(t, d)* can be calculated as:

$$\text{idf}(t, d) = \log \frac{n_d}{1+\text{df}(d,t)},$$

where n_d is the total number of documents, and *df(d, t)* is the number of documents *d* that contain the term *t*. Note that adding the constant 1 to the denominator is optional and serves the purpose of assigning a non-zero value to terms that occur in all training samples; the log is used to ensure that low document frequencies are not given too much weight.

Scikit-learn implements yet another transformer, the `TfidfTransformer` , that takes the raw term frequencies from `CountVectorizer` as input and transforms them into tf-idfs:

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf = TfidfTransformer(use_idf=True, norm='l2', smooth_idf=True
)
print(tfidf.fit_transform(count.fit_transform(docs)).toarray())
```

```
[[ 0.    0.43  0.    0.56  0.56  0.    0.43  0.    0.  ]
 [ 0.    0.43  0.    0.    0.    0.56  0.43  0.    0.56]
 [ 0.5   0.45  0.5   0.19  0.19  0.19  0.3   0.25  0.19]]
```

As we saw in the previous subsection, the word "is" (column 2) had the largest term frequency in the 3rd document, being the most frequently occurring word. However, after transforming the same feature vector into tf-idfs, we see that the word "is" is now associated with a relatively small tf-idf (0.45) in document 3 since it is also contained in documents 1 and 2 and thus is unlikely to contain any useful, discriminatory information.

However, if we'd manually calculated the tf-idfs of the individual terms in our feature vectors, we'd have noticed that the `TfidfTransformer` calculates the tf-idfs slightly differently compared to the standard textbook equations that we see earlier. The equations for the idf and tf-idf that were implemented in scikit-learn are:

$$\text{idf}(t,d) = log\frac{1+n_d}{1+\text{df}(d,t)}$$

The tf-idf equation that was implemented in scikit-learn is as follows:

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \times (\text{idf}(t,d)+1)$$

While it is also more typical to normalize the raw term frequencies before calculating the tf-idfs, the `TfidfTransformer` normalizes the tf-idfs directly.

By default (`norm='l2'`), scikit-learn's TfidfTransformer applies the L2-normalization, which returns a vector of length 1 by dividing an un-normalized feature vector *v* by its L2-norm:

$$v_{\text{norm}} = \frac{v}{||v||_2} = \frac{v}{\sqrt{v_1^2+v_2^2+\cdots+v_n^2}} = \frac{v}{\left(\sum_{i=1}^{n} v_i^2\right)^{\frac{1}{2}}}$$

To make sure that we understand how TfidfTransformer works, let us walk through an example and calculate the tf-idf of the word is in the 3rd document.

The word is has a term frequency of 3 (tf = 3) in document 3, and the document frequency of this term is 3 since the term is occurs in all three documents (df = 3). Thus, we can calculate the idf as follows:

$$\text{idf}(''\ is\ '', d3) = log\frac{1+3}{1+3} = 0$$

Now in order to calculate the tf-idf, we simply need to add 1 to the inverse document frequency and multiply it by the term frequency:

$$\text{tf-idf}("\ is\ ", d3) = 3 \times (0 + 1) = 3$$

```
tf_is = 3
n_docs = 3
idf_is = np.log((n_docs+1) / (3+1))
tfidf_is = tf_is * (idf_is + 1)
print('tf-idf of term "is" = %.2f' % tfidf_is)
```

```
tf-idf of term "is" = 3.00
```

If we repeated these calculations for all terms in the 3rd document, we'd obtain the following tf-idf vectors: [3.39, 3.0, 3.39, 1.29, 1.29, 1.29, 2.0 , 1.69, 1.29]. However, we notice that the values in this feature vector are different from the values that we obtained from the TfidfTransformer that we used previously. The step that we are missing in this tf-idf calculation is the L2-normalization, which can be applied as follows:

$$\text{tfi-df}_{norm} = \frac{[3.39, 3.0, 3.39, 1.29, 1.29, 1.29, 2.0, 1.69, 1.29]}{\sqrt{[3.39^2, 3.0^2, 3.39^2, 1.29^2, 1.29^2, 1.29^2, 2.0^2, 1.69^2, 1.29^2]}}$$

$$= [0.5, 0.45, 0.5, 0.19, 0.19, 0.19, 0.3, 0.25, 0.19]$$

$$\Rightarrow \text{tfi-df}_{norm}("\ is\ ", d3) = 0.45$$

```
tfidf = TfidfTransformer(use_idf=True, norm=None, smooth_idf=True
)
raw_tfidf = tfidf.fit_transform(count.fit_transform(docs)).toarr
ay()[-1]
raw_tfidf
```

```
array([ 3.39,  3.  ,  3.39,  1.29,  1.29,  1.29,  2.  ,  1.69,
1.29])
```

```
l2_tfidf = raw_tfidf / np.sqrt(np.sum(raw_tfidf**2))
l2_tfidf
```

```
array([ 0.5 ,  0.45,  0.5 ,  0.19,  0.19,  0.19,  0.3 ,  0.25,
0.19])
```

As we can see, the results match the results returned by scikit-learn's `TfidfTransformer` (below). Since we now understand how tf-idfs are calculated, let us proceed to the next sections and apply those concepts to the movie review dataset.

Cleaning text data

[back to top]

```
df.loc[0, 'review'][-50:]  # last 50 characters from the first d
ocument
```

```
u'is seven.<br /><br />Title (Brazil): Not Available'
```

text contains html markup tags, we need clean them. we will now remove all punctuation marks but only keep emoticon characters such as ":)"

```
import re #  Python 中处理正则表达式的模块
def preprocessor(text):
    text = re.sub(r'<[^>]*>', '', text)  # 去除 HTML 标签
    emoticons = re.findall(r'(?::|;|=)(?:-)?(?:\)|\(|D|P)', text
)
    text = re.sub(r'\W+', ' ', text.lower()) + \
           ' '.join(emoticons).replace('-', '')   # 将表情符号拼接
在正文后面
    return text
```

```
preprocessor(df.loc[0, 'review'][-50:])
```

```
u'is seven title brazil not available'
```

```
preprocessor("</a>This :) is :( a test :-)!")
```

```
'this is a test :) :( :)'
```

```
df['review'] = df['review'].apply(preprocessor)
```

Processing documents into tokens

[back to top]

split the text corpora into individual elements.

tokenize 是把长文本切成一系列单词，最简单的方式就是以 whitespace 切分.

word stemming 是将词转为最原始的形式, root form, (例如 running -> run), 一种算法是 Porter stemmer algorithm

以下需要使用 nltk, 需要先安装: `pip install nltk`

```
from nltk.stem.porter import PorterStemmer

porter = PorterStemmer()

def tokenizer(text):
    return text.split()  # 默认以空格切分
def tokenizer_porter(text):
    return [porter.stem(word) for word in text.split()]
```

```
tokenizer('runners like running and thus they run')
```

```
['runners', 'like', 'running', 'and', 'thus', 'they', 'run']
```

```
tokenizer_porter('runners like running and thus they run')
```

```
[u'runner', u'like', u'run', u'and', u'thu', u'they', u'run']
```

Stop-words (停词) 是最常见的一些单词, 它们的实际意义并不是很大, 大多是起辅助作用的, 但是它们的频次非常高, 所以需要去除, 例如 is, and, has

```
import nltk
nltk.download('stopwords')  # 下载停词
```

```
[nltk_data] Downloading package stopwords to /Users/alan/nltk_da
ta...
[nltk_data]   Package stopwords is already up-to-date!





True
```

```
from nltk.corpus import stopwords
stop = stopwords.words('english')
[w for w in tokenizer_porter('a runner likes running and runs a
lot')[-10:] if w not in stop]
```

```
[u'runner', u'like', u'run', u'run', u'lot']
```

Training a logistic regression model for sentiment classification

[back to top]

Strip HTML and punctuation to speed up the GridSearch later:

```
#  25,000 documents for training and 25,000 documents for testin
g, 需要大约40分钟
# 所以先使用5000 documents
X_train = df.loc[:5000, 'review'].values
y_train = df.loc[:5000, 'sentiment'].values
X_test = df.loc[5000:, 'review'].values
y_test = df.loc[5000:, 'sentiment'].values
```

```
from sklearn.grid_search import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer

# TfidfVectorizer 等同于 CountVectorizer + TfidfTransformer
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=None)

# grig search
param_grid = [{'vect__ngram_range': [(1,1)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer, tokenizer_porter],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [1.0, 10.0, 100.0]},
             {'vect__ngram_range': [(1,1)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer, tokenizer_porter],
               'vect__use_idf':[False],
               'vect__norm':[None],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [1.0, 10.0, 100.0]},
             ]

# 先转化为 tfidf matrix
lr_tfidf = Pipeline([('vect', tfidf),
                     ('clf', LogisticRegression(random_state=0))
])

gs_lr_tfidf = GridSearchCV(lr_tfidf, param_grid,
                           scoring='accuracy',
                           cv=5, verbose=1,
                           n_jobs=-1)
```

```
# 数据量减少为5000 后需要时间大约为8分钟
gs_lr_tfidf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits


[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  8.0min
[Parallel(n_jobs=-1)]: Done 240 out of 240 | elapsed: 10.5min fi
nished





GridSearchCV(cv=5, error_score='raise',
       estimator=Pipeline(steps=[('vect', TfidfVectorizer(analyz
er=u'word', binary=False, decode_error=u'strict',
        dtype=<type 'numpy.int64'>, encoding=u'utf-8', input=u'c
ontent',
        lowercase=False, max_df=1.0, max_features=None, min_df=1
,
        ngram_range=(1, 1), norm=u'l2', preprocessor=None, smoot
h_idf=Tru...nalty='l2', random_state=0, solver='liblinear', tol=
0.0001,
          verbose=0, warm_start=False))]),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid=[{'vect__ngram_range': [(1, 1)], 'vect__tokeni
zer': [<function tokenizer at 0x130dca410>, <function tokenizer_
porter at 0x130dca488>], 'clf__penalty': ['l1', 'l2'], 'clf__C':
 [1.0, 10.0, 100.0], 'vect__stop_words': [[u'i', u'me', u'my', u
'myself', u'we', u'our', u'ours', u'ourselves', u'y...x130dca488
>], 'vect__use_idf': [False], 'clf__C': [1.0, 10.0, 100.0], 'clf
__penalty': ['l1', 'l2']}],
       pre_dispatch='2*n_jobs', refit=True, scoring='accuracy',
verbose=1)
```

```
print('Best parameter set: %s ' % gs_lr_tfidf.best_params_)
print('CV Accuracy: %.3f' % gs_lr_tfidf.best_score_)
```

```
Best parameter set: {'vect__ngram_range': (1, 1), 'vect__tokeniz
er': <function tokenizer at 0x130dca410>, 'clf__penalty': 'l2',
'clf__C': 10.0, 'vect__stop_words': [u'i', u'me', u'my', u'mysel
f', u'we', u'our', u'ours', u'ourselves', u'you', u'your', u'you
rs', u'yourself', u'yourselves', u'he', u'him', u'his', u'himsel
f', u'she', u'her', u'hers', u'herself', u'it', u'its', u'itself
', u'they', u'them', u'their', u'theirs', u'themselves', u'what'
, u'which', u'who', u'whom', u'this', u'that', u'these', u'those
', u'am', u'is', u'are', u'was', u'were', u'be', u'been', u'bein
g', u'have', u'has', u'had', u'having', u'do', u'does', u'did',
u'doing', u'a', u'an', u'the', u'and', u'but', u'if', u'or', u'b
ecause', u'as', u'until', u'while', u'of', u'at', u'by', u'for',
 u'with', u'about', u'against', u'between', u'into', u'through',
 u'during', u'before', u'after', u'above', u'below', u'to', u'fr
om', u'up', u'down', u'in', u'out', u'on', u'off', u'over', u'un
der', u'again', u'further', u'then', u'once', u'here', u'there',
 u'when', u'where', u'why', u'how', u'all', u'any', u'both', u'e
ach', u'few', u'more', u'most', u'other', u'some', u'such', u'no
', u'nor', u'not', u'only', u'own', u'same', u'so', u'than', u't
oo', u'very', u's', u't', u'can', u'will', u'just', u'don', u'sh
ould', u'now', u'd', u'll', u'm', u'o', u're', u've', u'y', u'ai
n', u'aren', u'couldn', u'didn', u'doesn', u'hadn', u'hasn', u'h
aven', u'isn', u'ma', u'mightn', u'mustn', u'needn', u'shan', u'
shouldn', u'wasn', u'weren', u'won', u'wouldn']}
CV Accuracy: 0.862
```

```
clf = gs_lr_tfidf.best_estimator_
print('Test Accuracy: %.3f' % clf.score(X_test, y_test))
```

```
Test Accuracy: 0.873
```

Working with bigger data - online

algorithms and out-of-core learning

[back to top]

Out-of-Core learning is the task of training a machine learning model on a dataset that does not fit into memory or RAM. This requires the following conditions:

- a **feature extraction** layer with **fixed output dimensionality**
- knowing the list of all classes in advance (in this case we only have positive and negative tweets)
- a machine learning **algorithm that supports incremental learning** (the `partial_fit` method in scikit-learn).

```
import numpy as np
import re
from nltk.corpus import stopwords

stop = stopwords.words('english')

def tokenizer(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text.
lower())
    text = re.sub('[\W]+', ' ', text.lower()) + ' '.join(emotico
ns).replace('-', '')
    tokenized = [w for w in text.split() if w not in stop]
    return tokenized

def stream_docs(path):  # 生成器
    # generator funciton, reads in and returns one document at a
 time
    with open(path, 'r') as csv:
        next(csv) # skip header
        for line in csv:
            text, label = line[:-3], int(line[-2])
            yield text, label
```

```
# To verify that our stream_docs function works correctly
gen = stream_docs(path='./data/movie_data.csv')
next(gen)
```

```
('"In 1974, the teenager Martha Moxley (Maggie Grace) moves to t
he high-class area of Belle Haven, Greenwich, Connecticut. On th
e Mischief Night, eve of Halloween, she was murdered in the back
yard of her house and her murder remained unsolved. Twenty-two y
ears later, the writer Mark Fuhrman (Christopher Meloni), who is
 a former LA detective that has fallen in disgrace for perjury i
n O.J. Simpson trial and moved to Idaho, decides to investigate
the case with his partner Stephen Weeks (Andrew Mitchell) with t
he purpose of writing a book. The locals squirm and do not welco
me them, but with the support of the retired detective Steve Car
roll (Robert Forster) that was in charge of the investigation in
 the 70\'s, they discover the criminal and a net of power and mo
ney to cover the murder.<br /><br />""Murder in Greenwich"" is a
 good TV movie, with the true story of a murder of a fifteen yea
rs old girl that was committed by a wealthy teenager whose mothe
r was a Kennedy. The powerful and rich family used their influen
ce to cover the murder for more than twenty years. However, a sn
oopy detective and convicted perjurer in disgrace was able to di
sclose how the hideous crime was committed. The screenplay shows
 the investigation of Mark and the last days of Martha in parall
el, but there is a lack of the emotion in the dramatization. My
vote is seven.<br /><br />Title (Brazil): Not Available"',
 1)
```

```
def get_minibatch(doc_stream, size):
    '''
    take a document stream from the stream_docs function and
    return a particular number of documents
    '''
    docs, y = [], []
    try:
        for _ in range(size):
            text, label = next(doc_stream)
            docs.append(text)
            y.append(label)
    except StopIteration:
        return None, None
    return docs, y
```

```
from sklearn.feature_extraction.text import HashingVectorizer  # makes use of the Hashing trick
from sklearn.linear_model import SGDClassifier  # train a logistic regression model using small minibatches of documents

vect = HashingVectorizer(decode_error='ignore',
                         n_features=2**21,
                         preprocessor=None,
                         tokenizer=tokenizer)

clf = SGDClassifier(loss='log', random_state=1, n_iter=1)
doc_stream = stream_docs(path='./data/movie_data.csv')
```

```
# iterated over 45 minibatches of documents
# where each minibatch consists of 1,000 documents each
import pyprind
pbar = pyprind.ProgBar(45)

classes = np.array([0, 1])
for _ in range(45):
    X_train, y_train = get_minibatch(doc_stream, size=1000)
    if not X_train:
        break
    X_train = vect.transform(X_train)
    clf.partial_fit(X_train, y_train, classes=classes)
    pbar.update()
```

```
0%                            100%
[##############################] | ETA: 00:00:00
Total time elapsed: 00:00:40
```

```
#use the last 5,000 documents to evaluate the performance
X_test, y_test = get_minibatch(doc_stream, size=5000)
X_test = vect.transform(X_test)
print('Accuracy: %.3f' % clf.score(X_test, y_test))
```

```
Accuracy: 0.867
```

虽然准确率略低于前面，但训练速度快了很多，而且使用的内存更少

```
# use the last 5,000 documents to update our model
# 可以使用 partial_fit 继续训练
clf = clf.partial_fit(X_test, y_test)
```

Model persistence

训练模型是 expensive 并且耗费时间的，我们不希望在应用中每次都要重新训练模型，所以我们需要保存模型，并且能进行新的预测以及更新。 可以用到 pickle 模块来储存模型，将 python object 储存为byte code，可以读取也可以写入

[back to top]

After we trained the logistic regression model as shown above, we can save the classifier along with the stop words, Porter Stemmer, and `HashingVectorizer` as serialized objects to our local disk so that we can use the fitted classifier in our web application later.

```
import pickle
import os
#  created a movieclassifier directory
# created a pkl_objects subdirectory to save the serialized Pyth
on objects to our local drive
dest = os.path.join('movieclassifier', 'pkl_objects')
if not os.path.exists(dest):
    os.makedirs(dest)

# 写入
pickle.dump(stop, open(os.path.join(dest, 'stopwords.pkl'), 'wb'
), protocol=2)
pickle.dump(clf, open(os.path.join(dest, 'classifier.pkl'), 'wb'
), protocol=2)
```

Next, we save the `HashingVectorizer` as in a separate file so that we can import it later.

```
%%writefile movieclassifier/vectorizer.py
from sklearn.feature_extraction.text import HashingVectorizer
import re
import os
import pickle

cur_dir = os.path.dirname(__file__)
stop = pickle.load(open(
                os.path.join(cur_dir,
                'pkl_objects',
                'stopwords.pkl'), 'rb'))

def tokenizer(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)',
                           text.lower())
    text = re.sub('[\W]+', ' ', text.lower()) \
                   + ' '.join(emoticons).replace('-', '')
    tokenized = [w for w in text.split() if w not in stop]
    return tokenized

vect = HashingVectorizer(decode_error='ignore',
                         n_features=2**21,
                         preprocessor=None,
                         tokenizer=tokenizer)
```

```
Overwriting movieclassifier/vectorizer.py
```

After executing the preceeding code cells, we can now restart the IPython notebook kernel to check if the objects were serialized correctly.

First, change the current Python directory to `movieclassifer` :

```
import os
os.chdir('movieclassifier')
```

```
import pickle
import re
import os
from vectorizer import vect

clf = pickle.load(open(os.path.join('pkl_objects', 'classifier.p
kl'), 'rb'))
```

```
import numpy as np
label = {0:'negative', 1:'positive'}

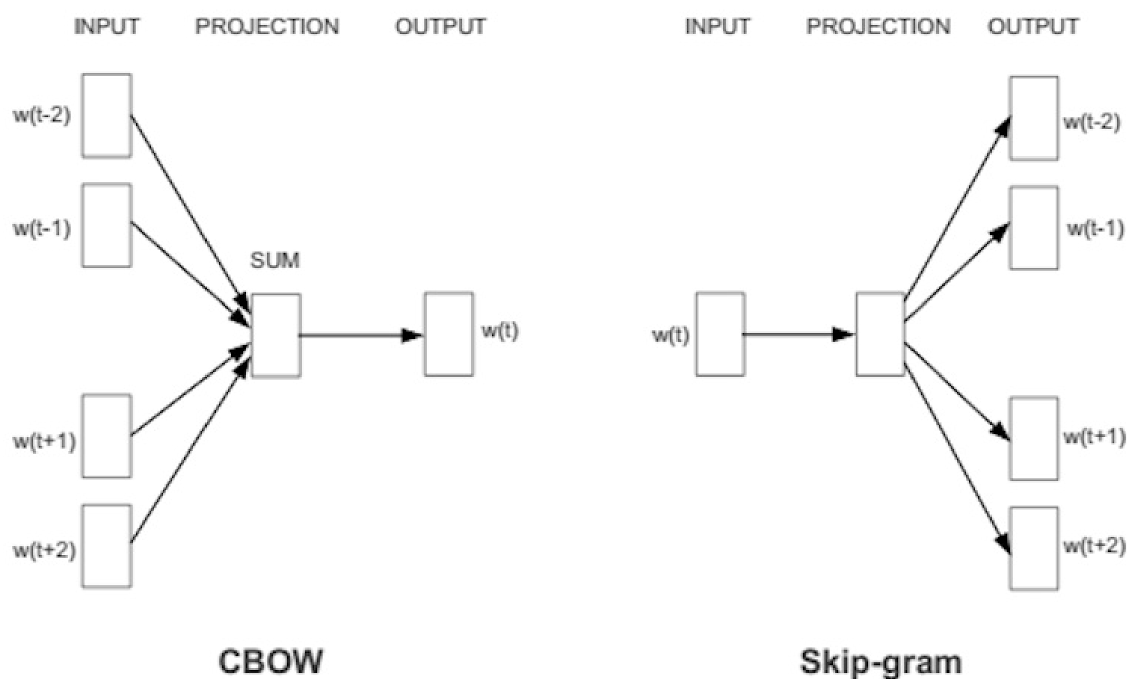
example = ['I love this movie']
X = vect.transform(example)
print('Prediction: %s\nProbability: %.2f%%' %\
      (label[clf.predict(X)[0]], np.max(clf.predict_proba(X))*100
))
```

```
Prediction: positive
Probability: 82.53%
```

word2vec

[back to top]

word2vec 是 Mikolov et al. 提出一种训练词向量的方法。它有 Continuous Bag-of-Words model (CBOW) and the Skip-Gram model 两种变式，前者是用一个词序列窗口中的其他词来预测中心词，后者则是用中心词来预测其他词。在实际使用 word2vec 时，一般使用 Skip-Gram model 结合 Negative Sampling 进行训练。




```
# gensim 库中包含了 word2vec 模块
from gensim.models.word2vec import Word2Vec
```

训练词向量， gensim 训练词向量时可以喂入一个 iterator

```
class MySentence:
    def __init__(self, data):
        self.data = data

    def __iter__(self):
        for line in self.data:
            yield line.lower().split()

train_corpus = MySentence(df['review'])

# 或者 train_corpus = [s.split() for s in df['review']]
```

```
model = Word2Vec(train_corpus,
                 size=200, # 词向量的维度
                 iter=20, # 数据在训练中用到的次数, 即 epoch 数
                 workers=4)  # 调用的进程数
```

简单查看词向量的结果

```
model.most_similar('good')
```

```
[(u'decent', 0.7367605566978455),
 (u'bad', 0.7155213356018066),
 (u'great', 0.7067341804504395),
 (u'nice', 0.6252745389938354),
 (u'cool', 0.5865251421928406),
 (u'passable', 0.5830472707748413),
 (u'funny', 0.5790721774101257),
 (u'fine', 0.5767843127250671),
 (u'lousy', 0.5747220516204834),
 (u'terrible', 0.5679782629013062)]
```

获得词对应的词向量

```
model['good']
```

```
array([-1.80578566,  2.59199047,  1.00538623,  0.00376823, -1.63
011074,
        1.19826996, -0.05748612,  0.34942579, -3.64253092,  0.77
269369,
       -0.12568648,  0.63311213, -0.29951221,  1.00349665,  0.57
139468,
        0.26391807, -2.15211248,  1.37436974,  0.72486091,  1.39
909697,
        1.39214003, -0.55878437,  1.28609359,  1.4021709 , -0.61
972415,
        1.40909946,  1.23801959,  0.12476239, -2.10969472, -0.16
599979,
        1.07056391, -0.9283669 , -1.24553549, -1.00224996,  0.85
817921,
       -2.98209238,  2.52684212, -0.84701031, -1.30793285, -3.52
102351,
        2.51759624,  1.55442834, -0.81624299,  0.80133152,  0.76
279849,
```

```
        -1.61152196,  0.84180117,  1.95387816,  0.39838204, -2.50
793004,
        -1.3055681 ,  2.4016645 ,  2.7259481 ,  1.65427065,  2.08
329916,
        -0.67130673, -2.77032781, -0.86488032, -0.83239168,  0.70
329827,
         0.1054525 , -0.48466596,  1.93771338,  0.57588094, -0.16
703451,
         2.06361747,  0.90677142,  1.05519104,  1.84931016,  0.77
389133,
         2.08386183, -2.16565847,  1.20824552,  0.05443871,  3.20
145893,
        -2.12376046, -0.46623436,  1.19470978, -0.60041159, -2.01
585841,
        -1.14179373,  0.25336841, -2.7429111 ,  0.19020027,  0.77
245057,
         1.07972777,  3.45747972, -1.17416704,  0.14808762, -0.04
948059,
         2.25656533, -1.87219381, -2.38383865, -2.15413165, -0.18
855318,
        -1.33702457, -0.72171617, -0.28952792,  0.9959411 ,  2.43
245673,
        -0.94892198,  1.32803464,  3.27445745,  0.44902089,  1.76
025999,
        -0.15520753, -0.42482886,  0.22728981, -0.93469167,  0.87
159711,
         1.20421445,  1.03957427,  0.56455994,  0.19218144, -1.84
992611,
         1.75870144, -2.13160086,  0.99597716, -1.92741001,  1.47
426069,
        -2.23345065, -0.38542071, -0.33888757, -0.50440568, -1.29
633522,
         1.55562842,  1.08055615,  1.2215786 , -1.75132465, -0.11
741698,
         0.77109945, -0.68743432, -0.31800482,  0.23696584,  0.77
463788,
        -1.75956166,  1.77490389,  0.07634074,  2.27028704,  0.66
218233,
         3.3049078 ,  3.30950427, -1.33130598, -0.92713892,  0.15
286015,
```

```
        0.287049  ,  0.42833641, -1.46425474,  0.9658314 ,  0.40
959406,
        0.47199422, -0.95312113, -3.23242712, -0.08304592, -1.21
38108 ,
        0.30118775, -0.96748334, -0.68403792, -2.61612749, -0.97
091049,
        0.03540362, -1.22498131,  0.23248808,  2.79292464,  1.39
140284,
        0.35824764,  0.53603202,  2.08097696,  1.61062503, -1.45
209515,
       -0.87486774,  0.6024124 ,  1.00253165, -0.25757971,  3.58
50575 ,
        0.37494364,  1.26284862, -1.56030715, -2.38174224, -2.99
708772,
        0.75272441,  0.95936358,  0.24002089,  1.7718761 ,  1.28
533518,
       -2.32952619,  2.59281707, -3.69943428,  0.34991279, -0.52
845728,
        1.35607052,  0.67054856, -1.21081388,  1.81272793,  0.64
519709,
       -0.4952068 ,  2.65413833,  1.53806341, -2.95830393,  2.38
203692], dtype=float32)
```

存储/读取模型

```
model.save('data/imdb.d2v')
```

```
model = Word2Vec.load('data/imdb.d2v')
```

利用训练好的词向量来进行情感分析

```
# 用 word vec 的均值作为 doc vec
def get_doc_vec(sentence, model):
    scores = [model[word] for word in sentence.split()
              if word in model]  # 如果词频小于 min_count, word2ve
c 不会把这个词放入 vocab 里

    return np.mean(scores, axis=0)
```

```
X_word2vec_train = np.array([get_doc_vec(sentence, model) for se
ntence in X_train])
X_word2vec_test = np.array([get_doc_vec(sentence, model) for sen
tence in X_test])
```

```
from sklearn.grid_search import GridSearchCV
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(random_state=0)

param_grid = [{'penalty': ['l1', 'l2'],
               'C': [1.0, 10.0, 100.0]}]

gs = GridSearchCV(lr, param_grid,
                  scoring='accuracy',
                  cv=5, verbose=1,
                  n_jobs=-1)

gs.fit(X_word2vec_train, y_train);
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits


[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:   14.5s fi
nished
```

```
gs.best_params_
```

```
{'C': 1.0, 'penalty': 'l2'}
```

```
gs.best_score_
```

```
0.85642871425714862
```

```
clf = gs.best_estimator_
print('Test Accuracy: %.3f' % clf.score(X_word2vec_test, y_test)
)
```

```
Test Accuracy: 0.869
```

中文新闻分类

```
from os import path
import os
import re
import pandas as pd
import numpy as np
```

```
rootdir = 'data/SogouC.reduced/Reduced'
dirs = os.listdir(rootdir)
dirs = [path.join(rootdir,f) for f in dirs if f.startswith('C')]
dirs
```

```
['data/SogouC.reduced/Reduced/C000008',
 'data/SogouC.reduced/Reduced/C000010',
 'data/SogouC.reduced/Reduced/C000013',
 'data/SogouC.reduced/Reduced/C000014',
 'data/SogouC.reduced/Reduced/C000016',
 'data/SogouC.reduced/Reduced/C000020',
 'data/SogouC.reduced/Reduced/C000022',
 'data/SogouC.reduced/Reduced/C000023',
 'data/SogouC.reduced/Reduced/C000024']
```

```
def load_txt(x):
    with open(x) as f:
        res = [t.decode('gbk','ignore') for t in f]
        return ''.join(res)
```

```
text_t = {}
for i, d in enumerate(dirs):
    files = os.listdir(d)
    files = [path.join(d, x) for x in files if x.endswith('txt')
and not x.startswith('.')]
    text_t[i] = [load_txt(f) for f in files]
```

```
print(text_t[0][0][:100])
```

```
    本报记者陈雪频实习记者唐翔发自上海
    一家刚刚成立两年的网络支付公司，它的目标是成为市值100亿美元的上市公司。
    这家公司叫做快钱，说这句话的是快钱的CEO关国光。他之前曾任网易的高级副
```

```
flen = [len(t) for t in text_t.values()]
labels = np.repeat(text_t.keys(),flen)
```

```
# flatter nested list
import itertools
merged = list(itertools.chain.from_iterable(text_t.values()))
```

```
df = pd.DataFrame({'label': labels, 'txt': merged})
df.head()
```

	label	txt
0	0	本报记者陈雪频实习记者唐翔发自上海\r\n 一家刚刚成立两年的网络支付公司，它的目标是...
1	0	证券通：百联股份未来5年有能力保持高速增长\r\n\r\n 深度报告 权威内参...
2	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....
3	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....
4	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....

```
# cut word
import jieba
jieba.enable_parallel(4)
def cutword_1(x):
    words = jieba.cut(x)
    return ' '.join(words)
```

```
Building prefix dict from the default dictionary ...
Loading model from cache /var/folders/y8/z6ws1f2907vbb33mcp8c633
00000gn/T/jieba.cache
Loading model cost 1.119 seconds.
Prefix dict has been built succesfully.
```

```
df['seg_word'] = df.txt.map(cutword_1)
```

```
df.head()
```

	label	txt	seg_word
0	0	本报记者陈雪频实习记者唐翔发自上海\r\n 一家刚刚成立两年的网络支付公司，它的目标是...	本报记者 陈雪频 实习 记者 唐翔 发自 上海 \r\n 一家 刚刚 成立 ...
1	0	证券通：百联股份未来5年有能力保持高速增长\r\n\r\n 深度报告 权威内参...	证券 通 ： 百联 股份 未来 5 年 有 能力 保持高速 增长 \r\n ...
2	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....	5 月 09 日 消息 快评 \r\n \r\n 深度 报告...
3	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....	5 月 09 日 消息 快评 \r\n \r\n 深度 报告...
4	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....	5 月 09 日 消息 快评 \r\n \r\n 深度 报告...

```
from cPickle import dump,load
dump(df, open('data/tmdf.pickle', 'wb'))
# df = load(open('df.pickle','rb'))
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(ngram_range=(1,1), min_df = 2, max_featur
es = 10000)
xvec = vect.fit_transform(df.seg_word)
xvec.shape
```

```
(17910, 10000)
```

```
y = df.label
```

```
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
train_X, test_X, train_y, test_y = train_test_split(xvec, y , tr
ain_size=0.7, random_state=1)
clf = MultinomialNB()
```

```
clf.fit(train_X, train_y)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
from sklearn import metrics
pre = clf.predict(test_X)
print metrics.classification_report(test_y, pre)
```

```
             precision    recall  f1-score   support

          0       0.90      0.88      0.89       577
          1       0.89      0.81      0.85       603
          2       0.88      0.82      0.85       619
          3       0.98      0.97      0.98       584
          4       0.86      0.88      0.87       570
          5       0.88      0.79      0.83       600
          6       0.77      0.90      0.83       600
          7       0.76      0.83      0.80       615
          8       0.92      0.93      0.93       605

avg / total       0.87      0.87      0.87      5373
```

```
# word2vec
txt = df.seg_word.values
txtlist = []
for sent in txt:
    temp = [w for w in sent.split()]
    txtlist.append(temp)
```

```
num_features = 100
min_word_count = 10
num_workers = 4
context = 5
epoch = 20
sample = 1e-5
```

```
from gensim.models import word2vec
```

```
model = word2vec.Word2Vec(txtlist, workers = num_workers,
                          sample = sample,
                          size = num_features,
                          min_count=min_word_count,
                          window = context,
                          iter = epoch)
```

```
model.syn0.shape
```

```
(57675, 100)
```

```
for w in model.most_similar(u'互联网'):
    print w[0], w[1]
```

```
网络 0.787674069405
门户网站 0.747487425804
搜索引擎 0.744884610176
无线 0.732329308987
B2B 0.713720798492
网络广告 0.712735056877
腾讯 0.702631175518
MSN 0.701346695423
大旗网 0.69608104229
Google 0.68867880106
```

```
#model.save('sogo_wv')
#model = word2vec.Word2Vec.load('sogo_wv')
```

```
# 将词向量平均化为文档向量
def sentvec_1(sent,m=num_features,model=model):
    res = np.zeros(m)
    words = sent.split()
    num = 0
    for w in words:
        if w in model.index2word:
            res += model[w]
            num += 1.0
    if num == 0: return np.zeros(m)
    else: return res/num
```

```
n = df.shape[0]
sent_matrix = np.zeros([n,num_features],float)
for i ,sent in enumerate(df.seg_word.values):
    sent_matrix[i,:] = sentvec_1(sent)
sent_matrix.shape
```

```
(17910, 100)
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.cross_validation import train_test_split
train_X, test_X, train_y, test_y = train_test_split(sent_matrix,
 y , train_size=0.7, random_state=1)
clf = GradientBoostingClassifier()
```

```
clf.fit(train_X, train_y)
from sklearn import metrics
pre = clf.predict(test_X)
print metrics.classification_report(test_y, pre)
```

```
             precision    recall  f1-score   support

          0       0.91      0.85      0.88       577
          1       0.84      0.82      0.83       603
          2       0.86      0.88      0.87       619
          3       0.98      0.97      0.98       584
          4       0.84      0.86      0.85       570
          5       0.84      0.80      0.82       600
          6       0.86      0.87      0.87       600
          7       0.77      0.83      0.80       615
          8       0.93      0.93      0.93       605

avg / total       0.87      0.87      0.87      5373
```

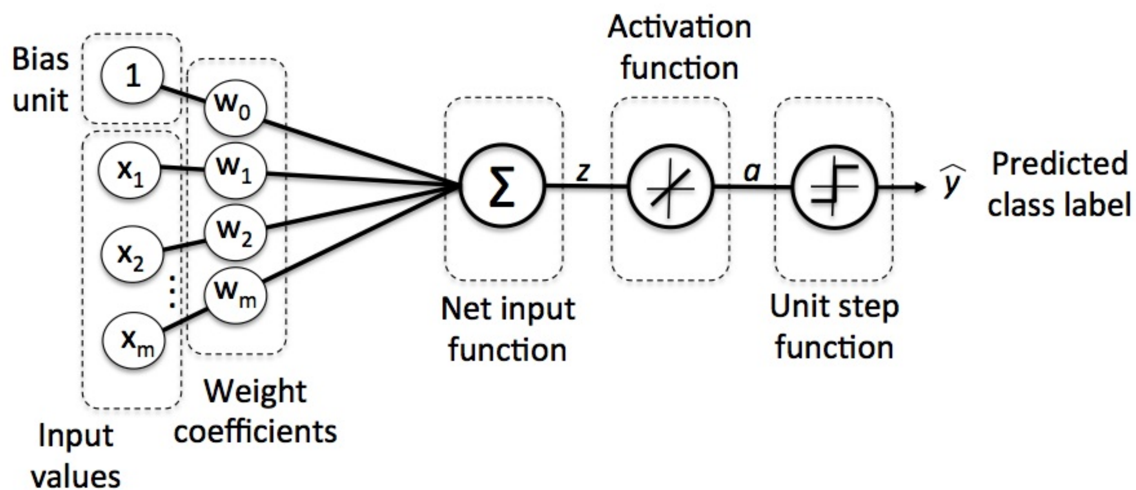
Sections

Modeling complex functions with artificial neural networks

[back to top]

Single-layer neural network recap

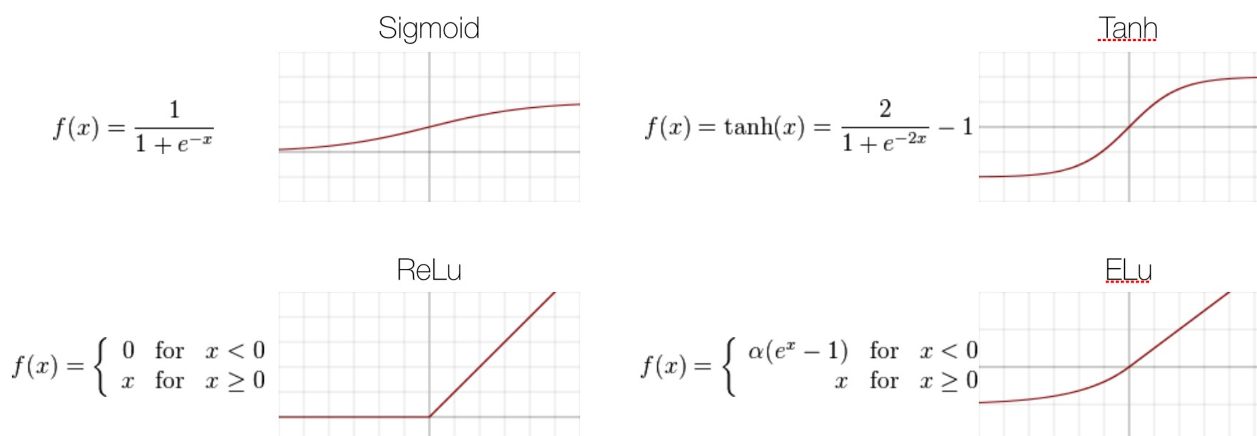
[back to top]



linear combination

Linear combination$z = \sum_{i=1}^{m} x_i \times w_i + w_0$，其中$w_0$是bias 项。

activation functions

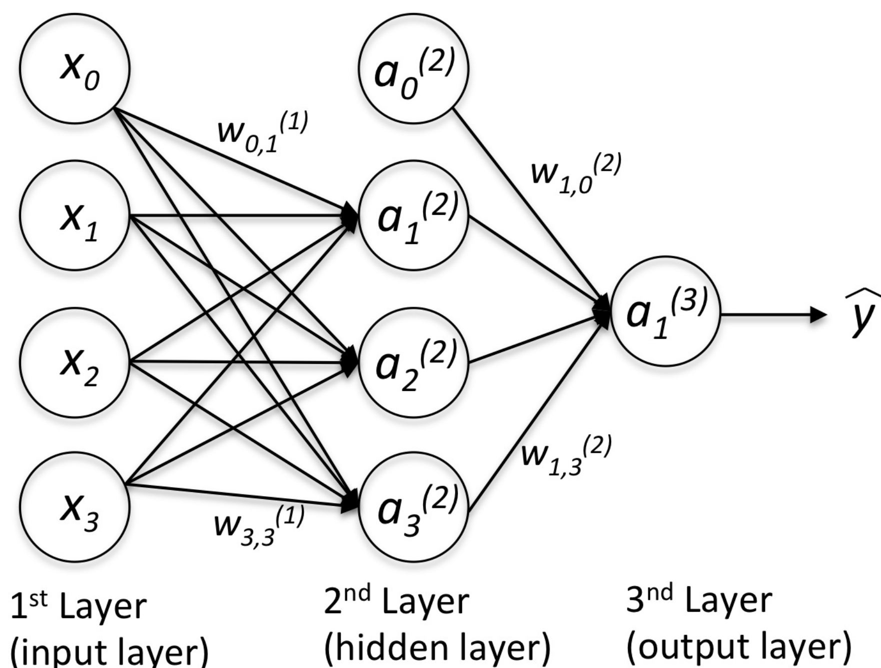


Introducing the multi-layer neural network architecture

[back to top]

Multi-layer perceptron (MLP)

MLP 可以看成多次线性转换和激活函数的堆叠，下图结构可以写成
$y = \sigma(\sigma(XW^{(1)} + b^{(1)})W^{(2)} + b^{(2)})$



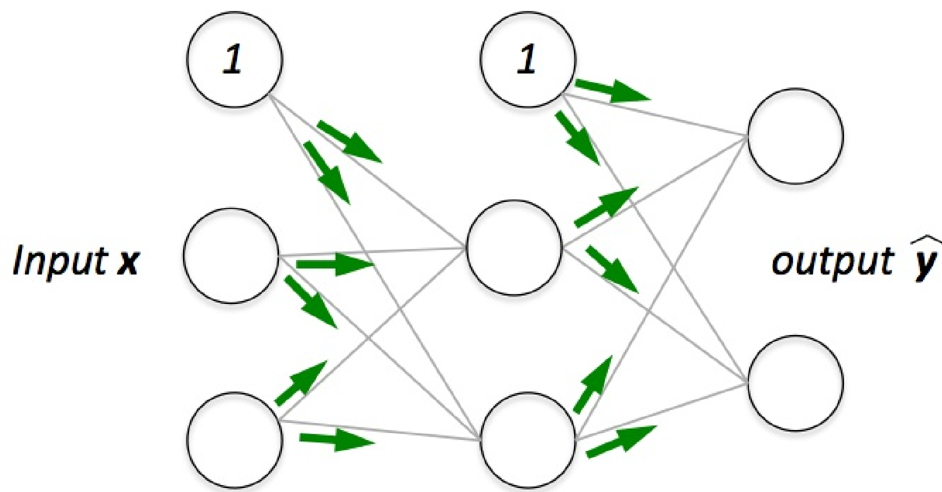
MLP learning procedure

[back to top]

1. Starting at the input layer, we forward propagate the patterns of the training data through the network to generate an output.
2. Based on the network's output, we calculate the error that we want to minimize using a cost function that we will describe later.
3. We backpropagate the error, find its derivative with respect to each weight in the network, and update the model.

Activating a neural network via forward propagation

[back to top]



Loss functions

[back to top]

$$\mathrm{MSE}=\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i-Y_i)^2$$

$$\mathrm{MAE}=\frac{1}{n}\sum_{i=1}^{n}|\hat{Y}_i-Y_i|$$

$$\mathrm{MAPE}=\frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{Y}_i-Y_i}{Y_i}\right|$$

$$\mathrm{MSLE}=\frac{1}{n}\sum_{i=1}^{n}(\log(\hat{Y}_i+1)-\log(Y_i+1))^2$$

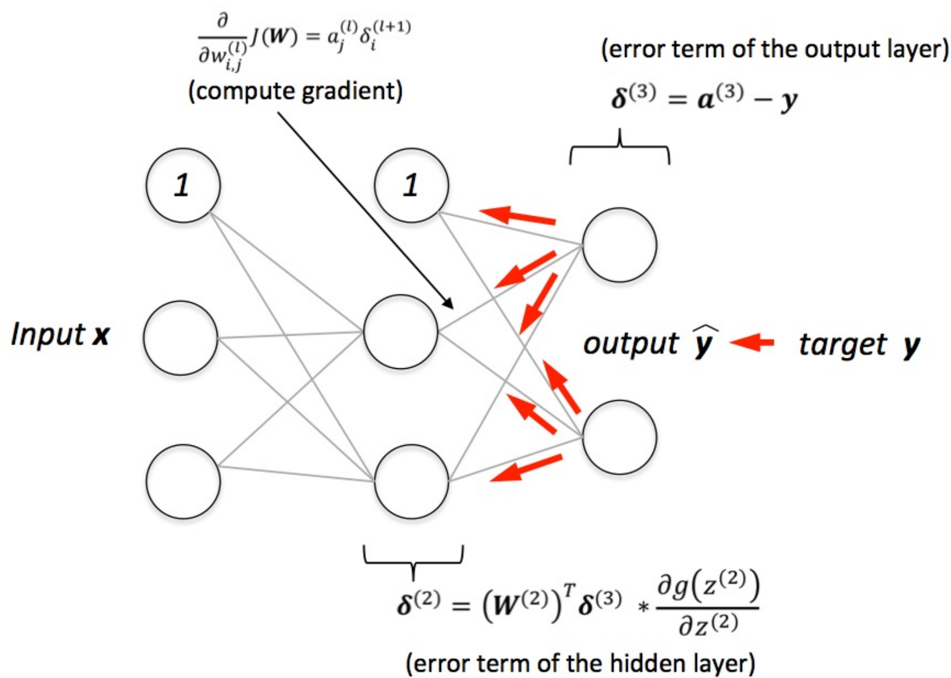
$$\mathrm{Hinge}=\frac{1}{n}\sum_{i=1}^{n}\max(1-Y_i\hat{Y}_i,0)$$

$$\mathrm{SH}=\frac{1}{n}\sum_{i=1}^{n}\left(\max(1-Y_i\hat{Y}_i,0)\right)^2$$

$$\mathrm{L}=-\frac{1}{n}\sum_{i=1}^{n}\left[Y_i\log\hat{Y}_i+(1-Y_i)\log\left(1-\hat{Y}_i\right)\right]$$

Training neural networks via backpropagation

[back to top]



Optimizaiton methods

[back to top]

Stochastic Gradient Descent (SGD)

$$\Delta\theta_{(k)} = -\alpha \sum_{i=1}^{n} \nabla L\left(x_i | \theta_{(k)}\right) + \mu \Delta\theta_{(k-1)}$$

- Learning rate α: how large a step to take
- Momentum μ: how important previous update is in calculating current update
- Decay: exponential rate of change of the learning rate as a function of the number of iteration

  at each iteration: $\alpha_{(k)} = \alpha_{(k-1)} / (1 + \text{decay} * k)$

Adam, Adagrad, Adamax, Adadelta, …

- Smoothing between steps
- Infer 2nd order information about optimization problem, like curvature
- Adaptive optimization algorithms adapt to the landscape and vary the parameters accordingly, performing parameterized scheduling with no human involvement

Classifying handwritten digits

Obtaining the MNIST dataset

[back to top]

The MNIST dataset is publicly available at http://yann.lecun.com/exdb/mnist/ and consists of the following four parts:

- Training set images: train-images-idx3-ubyte.gz (9.9 MB, 47 MB unzipped, 60,000 samples)
- Training set labels: train-labels-idx1-ubyte.gz (29 KB, 60 KB unzipped, 60,000 labels)
- Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 7.8 MB, 10,000 samples)
- Test set labels: t10k-labels-idx1-ubyte.gz (5 KB, 10 KB unzipped, 10,000 labels)

In this section, we will only be working with a subset of MNIST, thus, we only need to download the training set images and training set labels. After downloading the files, I recommend unzipping the files using the Unix/Linux gzip tool from the terminal for efficiency, e.g., using the command

```
gzip *ubyte.gz -d
```

in your local MNIST download directory, or, using your favorite unzipping tool if you are working with a machine running on Microsoft Windows. The images are stored in byte form, and using the following function, we will read them into NumPy arrays that we will use to train our MLP.

```
import os
import struct
import numpy as np

def load_mnist(path, kind='train'):
    """Load MNIST data from `path`"""
    labels_path = os.path.join(path,
                               '%s-labels-idx1-ubyte'
                                % kind)
    images_path = os.path.join(path,
                               '%s-images-idx3-ubyte'
                               % kind)

    with open(labels_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II',
                                 lbpath.read(8))
        labels = np.fromfile(lbpath,
                             dtype=np.uint8)

    with open(images_path, 'rb') as imgpath:
        magic, num, rows, cols = struct.unpack(">IIII",
                                               imgpath.read(16))
        images = np.fromfile(imgpath,
                             dtype=np.uint8).reshape(len(labels)
, 784)

    return images, labels
```

```
X_train, y_train = load_mnist('data/mnist', kind='train')
print('Rows: %d, columns: %d' % (X_train.shape[0], X_train.shape[
1]))
```

```
Rows: 60000, columns: 784
```

```
X_test, y_test = load_mnist('data/mnist', kind='t10k')
print('Rows: %d, columns: %d' % (X_test.shape[0], X_test.shape[1
]))
```

```
Rows: 10000, columns: 784
```

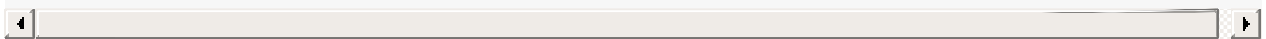
Visualize the first digit of each class:

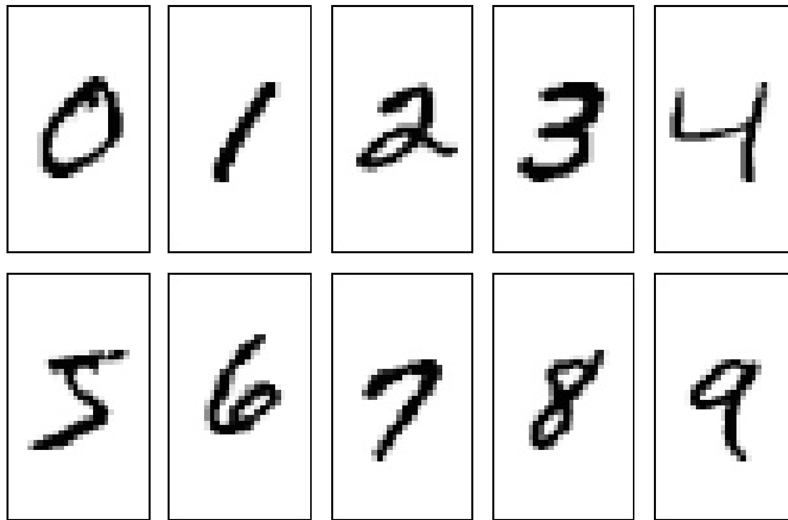
```
# import pandas as pd
# train = pd.read_csv('data/digit_recognizer/train.csv')
# train.shape
```

```
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(nrows=2, ncols=5, sharex=True, sharey=True
,)
ax = ax.flatten()
for i in range(10):
    img = X_train[y_train == i][0].reshape(28, 28)
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_all.png', dpi=300)
```

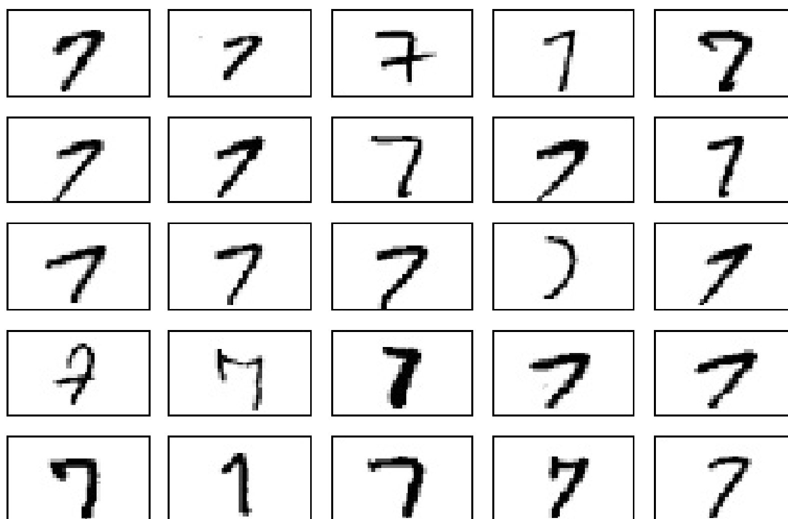




Visualize 25 different versions of "7":

```
fig, ax = plt.subplots(nrows=5, ncols=5, sharex=True, sharey=True
,)
ax = ax.flatten()
for i in range(25):
    img = X_train[y_train == 7][i].reshape(28, 28)
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_7.png', dpi=300)
```



Uncomment the following lines to optionally save the data in CSV format. However, note that those CSV files will take up a substantial amount of storage space:

- train_img.csv 1.1 GB (gigabytes)
- train_labels.csv 1.4 MB (megabytes)
- test_img.csv 187.0 MB
- test_labels 144 KB (kilobytes)

```
# np.savetxt('train_img.csv', X_train, fmt='%i', delimiter=',')
# np.savetxt('train_labels.csv', y_train, fmt='%i', delimiter=',
')
# X_train = np.genfromtxt('train_img.csv', dtype=int, delimiter=
',')
# y_train = np.genfromtxt('train_labels.csv', dtype=int, delimit
er=',')

# np.savetxt('test_img.csv', X_test, fmt='%i', delimiter=',')
# np.savetxt('test_labels.csv', y_test, fmt='%i', delimiter=',')
# X_test = np.genfromtxt('test_img.csv', dtype=int, delimiter=',
')
# y_test = np.genfromtxt('test_labels.csv', dtype=int, delimiter
=',')
```

Implementing a multi-layer perceptron

[back to top]

```
import numpy as np
from scipy.special import expit
import sys


class NeuralNetMLP(object):
    """ Feedforward neural network / Multi-layer perceptron clas
```

```
sifier.

    Parameters
    ------------
    n_output : int
      Number of output units, should be equal to the
      number of unique class labels.

    n_features : int
      Number of features (dimensions) in the target dataset.
      Should be equal to the number of columns in the X array.

    n_hidden : int (default: 30)
      Number of hidden units.

    l1 : float (default: 0.0)
      Lambda value for L1-regularization.
      No regularization if l1=0.0 (default)

    l2 : float (default: 0.0)
      Lambda value for L2-regularization.
      No regularization if l2=0.0 (default)

    epochs : int (default: 500)
      Number of passes over the training set.

    eta : float (default: 0.001)
      Learning rate.

    alpha : float (default: 0.0)
      Momentum constant. Factor multiplied with the
      gradient of the previous epoch t-1 to improve
      learning speed
      w(t) := w(t) - (grad(t) + alpha*grad(t-1))

    decrease_const : float (default: 0.0)
      Decrease constant. Shrinks the learning rate
      after each epoch via eta / (1 + epoch*decrease_const)

    shuffle : bool (default: False)
```

```
        Shuffles training data every epoch if True to prevent circ
les.

    minibatches : int (default: 1)
        Divides training data into k minibatches for efficiency.
        Normal gradient descent learning if k=1 (default).

    random_state : int (default: None)
        Set random state for shuffling and initializing the weight
s.

    Attributes
    -----------
    cost_ : list
        Sum of squared errors after each epoch.

    """
    def __init__(self, n_output, n_features, n_hidden=30,
                 l1=0.0, l2=0.0, epochs=500, eta=0.001,
                 alpha=0.0, decrease_const=0.0, shuffle=True,
                 minibatches=1, random_state=None):

        np.random.seed(random_state)
        self.n_output = n_output
        self.n_features = n_features
        self.n_hidden = n_hidden
        self.w1, self.w2 = self._initialize_weights()
        self.l1 = l1
        self.l2 = l2
        self.epochs = epochs
        self.eta = eta
        self.alpha = alpha
        self.decrease_const = decrease_const
        self.shuffle = shuffle
        self.minibatches = minibatches

    def _encode_labels(self, y, k):
        """Encode labels into one-hot representation

        Parameters
```

```
        ------------
        y : array, shape = [n_samples]
            Target values.

        Returns
        -----------
        onehot : array, shape = (n_labels, n_samples)

        """
        onehot = np.zeros((k, y.shape[0]))
        for idx, val in enumerate(y):
            onehot[val, idx] = 1.0
        return onehot

    def _initialize_weights(self):
        """Initialize weights with small random numbers."""
        w1 = np.random.uniform(-1.0, 1.0, size=self.n_hidden*(se
lf.n_features + 1))
        w1 = w1.reshape(self.n_hidden, self.n_features + 1)
        w2 = np.random.uniform(-1.0, 1.0, size=self.n_output*(se
lf.n_hidden + 1))
        w2 = w2.reshape(self.n_output, self.n_hidden + 1)
        return w1, w2

    def _sigmoid(self, z):
        """Compute logistic function (sigmoid)

        Uses scipy.special.expit to avoid overflow
        error for very small input values z.

        """
        # return 1.0 / (1.0 + np.exp(-z))
        return expit(z)

    def _sigmoid_gradient(self, z):
        """Compute gradient of the logistic function"""
        sg = self._sigmoid(z)
        return sg * (1 - sg)  # sigmoid 函数的导数比较简单，就是 sg
* (1-sg)
```

```
    def _add_bias_unit(self, X, how='column'):
        """Add bias unit (column or row of 1s) to array at index
 0"""
        if how == 'column':
            X_new = np.ones((X.shape[0], X.shape[1]+1))
            X_new[:, 1:] = X
        elif how == 'row':
            X_new = np.ones((X.shape[0]+1, X.shape[1]))
            X_new[1:, :] = X
        else:
            raise AttributeError('`how` must be `column` or `row
`')
        return X_new

    def _feedforward(self, X, w1, w2):
        """Compute feedforward step

        Parameters
        -----------
        X : array, shape = [n_samples, n_features]
          Input layer with original features.

        w1 : array, shape = [n_hidden_units, n_features]
          Weight matrix for input layer -> hidden layer.

        w2 : array, shape = [n_output_units, n_hidden_units]
          Weight matrix for hidden layer -> output layer.

        Returns
        ----------
        a1 : array, shape = [n_samples, n_features+1]
          Input values with bias unit.

        z2 : array, shape = [n_hidden, n_samples]
          Net input of hidden layer.

        a2 : array, shape = [n_hidden+1, n_samples]
          Activation of hidden layer.

        z3 : array, shape = [n_output_units, n_samples]
```

```
          Net input of output layer.

        a3 : array, shape = [n_output_units, n_samples]
          Activation of output layer.

        """
        a1 = self._add_bias_unit(X, how='column')
        z2 = w1.dot(a1.T)
        a2 = self._sigmoid(z2)
        a2 = self._add_bias_unit(a2, how='row')
        z3 = w2.dot(a2)
        a3 = self._sigmoid(z3)
        return a1, z2, a2, z3, a3

    def _L2_reg(self, lambda_, w1, w2):
        """Compute L2-regularization cost"""
        return (lambda_/2.0) * (np.sum(w1[:, 1:] ** 2) + np.sum(
w2[:, 1:] ** 2))

    def _L1_reg(self, lambda_, w1, w2):
        """Compute L1-regularization cost"""
        return (lambda_/2.0) * (np.abs(w1[:, 1:]).sum() + np.abs
(w2[:, 1:]).sum())

    def _get_cost(self, y_enc, output, w1, w2):
        """Compute cost function.

        y_enc : array, shape = (n_labels, n_samples)
          one-hot encoded class labels.

        output : array, shape = [n_output_units, n_samples]
          Activation of the output layer (feedforward)

        w1 : array, shape = [n_hidden_units, n_features]
          Weight matrix for input layer -> hidden layer.

        w2 : array, shape = [n_output_units, n_hidden_units]
          Weight matrix for hidden layer -> output layer.

        Returns
```

```
        ---------
        cost : float
          Regularized cost.

        """
        term1 = -y_enc * (np.log(output))
        term2 = (1 - y_enc) * np.log(1 - output)
        cost = np.sum(term1 - term2)  # 交互熵
        L1_term = self._L1_reg(self.l1, w1, w2)
        L2_term = self._L2_reg(self.l2, w1, w2)
        cost = cost + L1_term + L2_term
        return cost

    def _get_gradient(self, a1, a2, a3, z2, y_enc, w1, w2):
        """ Compute gradient step using backpropagation.

        Parameters
        ------------
        a1 : array, shape = [n_samples, n_features+1]
          Input values with bias unit.

        a2 : array, shape = [n_hidden+1, n_samples]
          Activation of hidden layer.

        a3 : array, shape = [n_output_units, n_samples]
          Activation of output layer.

        z2 : array, shape = [n_hidden, n_samples]
          Net input of hidden layer.

        y_enc : array, shape = (n_labels, n_samples)
          one-hot encoded class labels.

        w1 : array, shape = [n_hidden_units, n_features]
          Weight matrix for input layer -> hidden layer.

        w2 : array, shape = [n_output_units, n_hidden_units]
          Weight matrix for hidden layer -> output layer.

        Returns
```

```
        ---------

        grad1 : array, shape = [n_hidden_units, n_features]
          Gradient of the weight matrix w1.

        grad2 : array, shape = [n_output_units, n_hidden_units]
            Gradient of the weight matrix w2.

        """
        # backpropagation
        sigma3 = a3 - y_enc
        z2 = self._add_bias_unit(z2, how='row')
        sigma2 = w2.T.dot(sigma3) * self._sigmoid_gradient(z2)
        sigma2 = sigma2[1:, :]
        grad1 = sigma2.dot(a1)
        grad2 = sigma3.dot(a2.T)

        # regularize
        grad1[:, 1:] += (w1[:, 1:] * (self.l1 + self.l2))
        grad2[:, 1:] += (w2[:, 1:] * (self.l1 + self.l2))

        return grad1, grad2

    def predict(self, X):
        """Predict class labels

        Parameters
        -----------
        X : array, shape = [n_samples, n_features]
          Input layer with original features.

        Returns:
        ----------
        y_pred : array, shape = [n_samples]
          Predicted class labels.

        """
        if len(X.shape) != 2:
            raise AttributeError('X must be a [n_samples, n_feat
ures] array.\n'
```

```
                                 'Use X[:,None] for 1-feature cl
assification,'
                                 '\nor X[[i]] for 1-sample class
ification')

        a1, z2, a2, z3, a3 = self._feedforward(X, self.w1, self.
w2)
        y_pred = np.argmax(z3, axis=0)
        return y_pred

    def fit(self, X, y, print_progress=False):
        """ Learn weights from training data.

        Parameters
        -----------
        X : array, shape = [n_samples, n_features]
          Input layer with original features.

        y : array, shape = [n_samples]
          Target class labels.

        print_progress : bool (default: False)
          Prints progress as the number of epochs
          to stderr.

        Returns:
        ----------
        self

        """
        self.cost_ = []
        X_data, y_data = X.copy(), y.copy()
        y_enc = self._encode_labels(y, self.n_output)

        delta_w1_prev = np.zeros(self.w1.shape)
        delta_w2_prev = np.zeros(self.w2.shape)

        for i in range(self.epochs):

            # adaptive learning rate
```

```
            self.eta /= (1 + self.decrease_const*i)

            if print_progress:
                sys.stderr.write('\rEpoch: %d/%d' % (i+1, self.e
pochs))
                sys.stderr.flush()

            if self.shuffle:
                idx = np.random.permutation(y_data.shape[0])
                X_data, y_data = X_data[idx], y_data[idx]

            mini = np.array_split(range(y_data.shape[0]), self.m
inibatches)
            for idx in mini:

                # feedforward
                a1, z2, a2, z3, a3 = self._feedforward(X[idx], s
elf.w1, self.w2)
                cost = self._get_cost(y_enc=y_enc[:, idx],
                                      output=a3,
                                      w1=self.w1,
                                      w2=self.w2)
                self.cost_.append(cost)

                # compute gradient via backpropagation
                grad1, grad2 = self._get_gradient(a1=a1, a2=a2,
                                                  a3=a3, z2=z2,
                                                  y_enc=y_enc[:,
 idx],

                                                  w1=self.w1,
                                                  w2=self.w2)

                delta_w1, delta_w2 = self.eta * grad1, self.eta
* grad2
                self.w1 -= (delta_w1 + (self.alpha * delta_w1_pr
ev))
                self.w2 -= (delta_w2 + (self.alpha * delta_w2_pr
ev))
                delta_w1_prev, delta_w2_prev = delta_w1, delta_w
2
```

```
        return self
```

Training an artificial neural network

[back to top]

```
nn = NeuralNetMLP(n_output=10,
                  n_features=X_train.shape[1],
                  n_hidden=50,
                  l2=0.1,
                  l1=0.0,
                  epochs=1000,
                  eta=0.001,
                  alpha=0.001,
                  decrease_const=0.00001,
                  minibatches=50,
                  random_state=1)
```

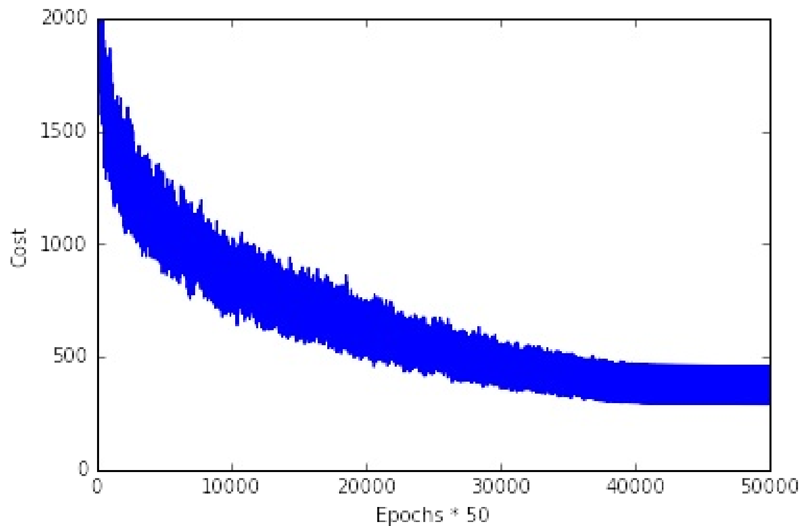
```
nn.fit(X_train, y_train, print_progress=True)
```

```
Epoch: 1000/1000



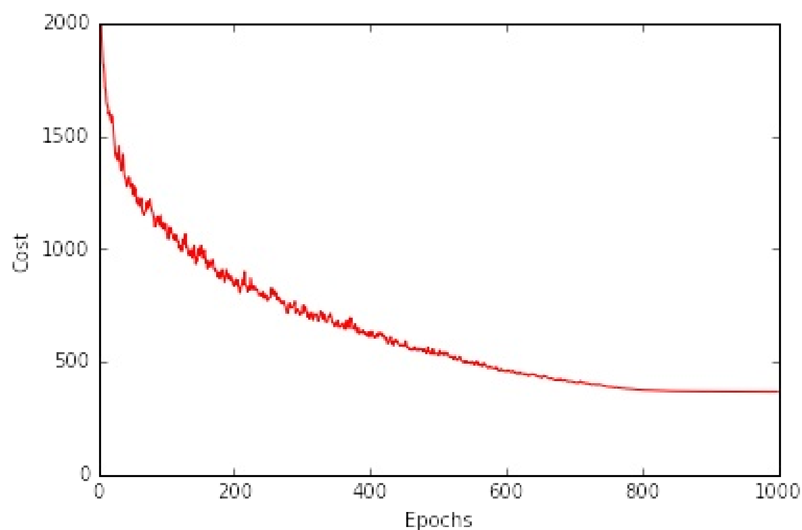

<__main__.NeuralNetMLP at 0x11854e7d0>
```

```
plt.plot(range(len(nn.cost_)), nn.cost_)
plt.ylim([0, 2000])
plt.ylabel('Cost')
plt.xlabel('Epochs * 50')
plt.tight_layout()
# plt.savefig('./figures/cost.png', dpi=300)
```




```
batches = np.array_split(range(len(nn.cost_)), 1000)
cost_ary = np.array(nn.cost_)
cost_avgs = [np.mean(cost_ary[i]) for i in batches]
```

```
plt.plot(range(len(cost_avgs)), cost_avgs, color='red')
plt.ylim([0, 2000])
plt.ylabel('Cost')
plt.xlabel('Epochs')
plt.tight_layout();
#plt.savefig('./figures/cost2.png', dpi=300)
```




```
y_train_pred = nn.predict(X_train)
acc = np.sum(y_train == y_train_pred, axis=0) / X_train.shape[0]
print('Training accuracy: %.2f%%' % (acc * 100))
```

```
Training accuracy: 0.00%
```

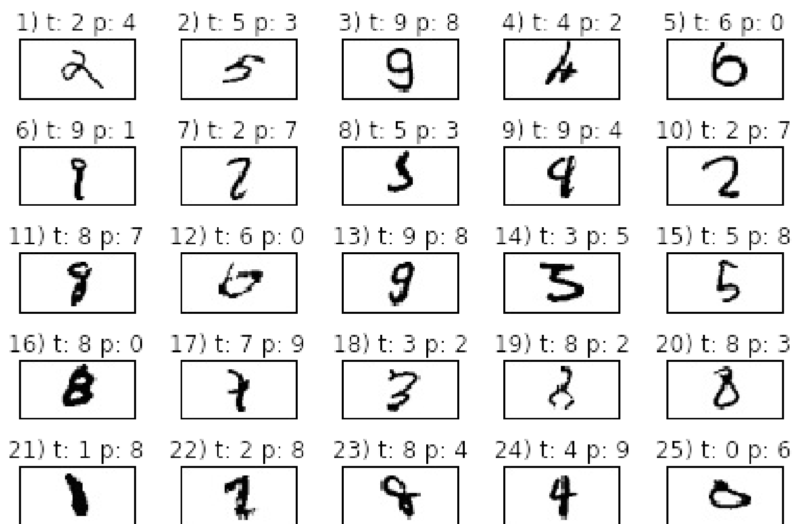
```
y_test_pred = nn.predict(X_test)
acc = np.sum(y_test == y_test_pred, axis=0) / X_test.shape[0]
print('Training accuracy: %.2f%%' % (acc * 100))
```

```
Training accuracy: 0.00%
```

```
miscl_img = X_test[y_test != y_test_pred][:25]
correct_lab = y_test[y_test != y_test_pred][:25]
miscl_lab= y_test_pred[y_test != y_test_pred][:25]

fig, ax = plt.subplots(nrows=5, ncols=5, sharex=True, sharey=True
,)
ax = ax.flatten()
for i in range(25):
    img = miscl_img[i].reshape(28, 28)
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')
    ax[i].set_title('%d) t: %d p: %d' % (i+1, correct_lab[i], mi
scl_lab[i]))

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_miscl.png', dpi=300)
plt.show()
```



Debugging neural networks with gradient checking

[back to top]

```
import numpy as np
from scipy.special import expit
import sys


class MLPGradientCheck(object):
    """ Feedforward neural network / Multi-layer perceptron clas
sifier.

    Parameters
    ------------
    n_output : int
        Number of output units, should be equal to the
        number of unique class labels.
    n_features : int
        Number of features (dimensions) in the target dataset.
        Should be equal to the number of columns in the X array.
    n_hidden : int (default: 30)
        Number of hidden units.
    l1 : float (default: 0.0)
        Lambda value for L1-regularization.
        No regularization if l1=0.0 (default)
    l2 : float (default: 0.0)
        Lambda value for L2-regularization.
        No regularization if l2=0.0 (default)
    epochs : int (default: 500)
        Number of passes over the training set.
    eta : float (default: 0.001)
        Learning rate.
    alpha : float (default: 0.0)
        Momentum constant. Factor multiplied with the
        gradient of the previous epoch t-1 to improve
        learning speed
        w(t) := w(t) - (grad(t) + alpha*grad(t-1))
    decrease_const : float (default: 0.0)
        Decrease constant. Shrinks the learning rate
        after each epoch via eta / (1 + epoch*decrease_const)
    shuffle : bool (default: False)
```

```
        Shuffles training data every epoch if True to prevent ci
rcles.
    minibatches : int (default: 1)
        Divides training data into k minibatches for efficiency.
        Normal gradient descent learning if k=1 (default).
    random_state : int (default: None)
        Set random state for shuffling and initializing the weig
hts.

    Attributes
    -----------
    cost_ : list
        Sum of squared errors after each epoch.

    """
    def __init__(self, n_output, n_features, n_hidden=30,
                 l1=0.0, l2=0.0, epochs=500, eta=0.001,
                 alpha=0.0, decrease_const=0.0, shuffle=True,
                 minibatches=1, random_state=None):

        np.random.seed(random_state)
        self.n_output = n_output
        self.n_features = n_features
        self.n_hidden = n_hidden
        self.w1, self.w2 = self._initialize_weights()
        self.l1 = l1
        self.l2 = l2
        self.epochs = epochs
        self.eta = eta
        self.alpha = alpha
        self.decrease_const = decrease_const
        self.shuffle = shuffle
        self.minibatches = minibatches

    def _encode_labels(self, y, k):
        """Encode labels into one-hot representation

        Parameters
        ------------
        y : array, shape = [n_samples]
```

```
            Target values.

        Returns
        -----------
        onehot : array, shape = (n_labels, n_samples)

        """
        onehot = np.zeros((k, y.shape[0]))
        for idx, val in enumerate(y):
            onehot[val, idx] = 1.0
        return onehot

    def _initialize_weights(self):
        """Initialize weights with small random numbers."""
        w1 = np.random.uniform(-1.0, 1.0,
                               size=self.n_hidden*(self.n_featur
es + 1))
        w1 = w1.reshape(self.n_hidden, self.n_features + 1)
        w2 = np.random.uniform(-1.0, 1.0,
                               size=self.n_output*(self.n_hidden
 + 1))
        w2 = w2.reshape(self.n_output, self.n_hidden + 1)
        return w1, w2

    def _sigmoid(self, z):
        """Compute logistic function (sigmoid)

        Uses scipy.special.expit to avoid overflow
        error for very small input values z.

        """
        # return 1.0 / (1.0 + np.exp(-z))
        return expit(z)

    def _sigmoid_gradient(self, z):
        """Compute gradient of the logistic function"""
        sg = self._sigmoid(z)
        return sg * (1.0 - sg)

    def _add_bias_unit(self, X, how='column'):
```

```
        """Add bias unit (column or row of 1s) to array at index
 0"""
        if how == 'column':
            X_new = np.ones((X.shape[0], X.shape[1] + 1))
            X_new[:, 1:] = X
        elif how == 'row':
            X_new = np.ones((X.shape[0]+1, X.shape[1]))
            X_new[1:, :] = X
        else:
            raise AttributeError('`how` must be `column` or `row
`')
        return X_new

    def _feedforward(self, X, w1, w2):
        """Compute feedforward step

        Parameters
        -----------
        X : array, shape = [n_samples, n_features]
            Input layer with original features.
        w1 : array, shape = [n_hidden_units, n_features]
            Weight matrix for input layer -> hidden layer.
        w2 : array, shape = [n_output_units, n_hidden_units]
            Weight matrix for hidden layer -> output layer.

        Returns
        ----------
        a1 : array, shape = [n_samples, n_features+1]
            Input values with bias unit.
        z2 : array, shape = [n_hidden, n_samples]
            Net input of hidden layer.
        a2 : array, shape = [n_hidden+1, n_samples]
            Activation of hidden layer.
        z3 : array, shape = [n_output_units, n_samples]
            Net input of output layer.
        a3 : array, shape = [n_output_units, n_samples]
            Activation of output layer.

        """
        a1 = self._add_bias_unit(X, how='column')
```

```
        z2 = w1.dot(a1.T)
        a2 = self._sigmoid(z2)
        a2 = self._add_bias_unit(a2, how='row')
        z3 = w2.dot(a2)
        a3 = self._sigmoid(z3)
        return a1, z2, a2, z3, a3

    def _L2_reg(self, lambda_, w1, w2):
        """Compute L2-regularization cost"""
        return (lambda_/2.0) * (np.sum(w1[:, 1:] ** 2) +
                                np.sum(w2[:, 1:] ** 2))

    def _L1_reg(self, lambda_, w1, w2):
        """Compute L1-regularization cost"""
        return (lambda_/2.0) * (np.abs(w1[:, 1:]).sum() +
                                np.abs(w2[:, 1:]).sum())

    def _get_cost(self, y_enc, output, w1, w2):
        """Compute cost function.

        Parameters
        ----------
        y_enc : array, shape = (n_labels, n_samples)
            one-hot encoded class labels.
        output : array, shape = [n_output_units, n_samples]
            Activation of the output layer (feedforward)
        w1 : array, shape = [n_hidden_units, n_features]
            Weight matrix for input layer -> hidden layer.
        w2 : array, shape = [n_output_units, n_hidden_units]
            Weight matrix for hidden layer -> output layer.

        Returns
        ---------
        cost : float
            Regularized cost.

        """
        term1 = -y_enc * (np.log(output))
        term2 = (1.0 - y_enc) * np.log(1.0 - output)
        cost = np.sum(term1 - term2)
```

```
        L1_term = self._L1_reg(self.l1, w1, w2)
        L2_term = self._L2_reg(self.l2, w1, w2)
        cost = cost + L1_term + L2_term
        return cost

    def _get_gradient(self, a1, a2, a3, z2, y_enc, w1, w2):
        """ Compute gradient step using backpropagation.

        Parameters
        ------------
        a1 : array, shape = [n_samples, n_features+1]
            Input values with bias unit.
        a2 : array, shape = [n_hidden+1, n_samples]
            Activation of hidden layer.
        a3 : array, shape = [n_output_units, n_samples]
            Activation of output layer.
        z2 : array, shape = [n_hidden, n_samples]
            Net input of hidden layer.
        y_enc : array, shape = (n_labels, n_samples)
            one-hot encoded class labels.
        w1 : array, shape = [n_hidden_units, n_features]
            Weight matrix for input layer -> hidden layer.
        w2 : array, shape = [n_output_units, n_hidden_units]
            Weight matrix for hidden layer -> output layer.

        Returns
        ---------
        grad1 : array, shape = [n_hidden_units, n_features]
            Gradient of the weight matrix w1.
        grad2 : array, shape = [n_output_units, n_hidden_units]
            Gradient of the weight matrix w2.

        """
        # backpropagation
        sigma3 = a3 - y_enc
        z2 = self._add_bias_unit(z2, how='row')
        sigma2 = w2.T.dot(sigma3) * self._sigmoid_gradient(z2)
        sigma2 = sigma2[1:, :]
        grad1 = sigma2.dot(a1)
        grad2 = sigma3.dot(a2.T)
```

```
        # regularize
        grad1[:, 1:] += (w1[:, 1:] * (self.l1 + self.l2))
        grad2[:, 1:] += (w2[:, 1:] * (self.l1 + self.l2))

        return grad1, grad2

    def _gradient_checking(self, X, y_enc, w1, w2, epsilon, grad
1, grad2):
        """ Apply gradient checking (for debugging only)

        Returns
        ---------
        relative_error : float
          Relative error between the numerically
          approximated gradients and the backpropagated gradient
s.

        """
        num_grad1 = np.zeros(np.shape(w1))
        epsilon_ary1 = np.zeros(np.shape(w1))
        for i in range(w1.shape[0]):
            for j in range(w1.shape[1]):
                epsilon_ary1[i, j] = epsilon
                a1, z2, a2, z3, a3 = self._feedforward(X,
                                                       w1 - epsi
lon_ary1, w2)
                cost1 = self._get_cost(y_enc, a3, w1-epsilon_ary
1, w2)
                a1, z2, a2, z3, a3 = self._feedforward(X,
                                                       w1 + epsi
lon_ary1, w2)
                cost2 = self._get_cost(y_enc, a3, w1 + epsilon_a
ry1, w2)
                num_grad1[i, j] = (cost2 - cost1) / (2.0 * epsil
on)
                epsilon_ary1[i, j] = 0

        num_grad2 = np.zeros(np.shape(w2))
        epsilon_ary2 = np.zeros(np.shape(w2))
```

```
        for i in range(w2.shape[0]):
            for j in range(w2.shape[1]):
                epsilon_ary2[i, j] = epsilon
                a1, z2, a2, z3, a3 = self._feedforward(X, w1,
                                                       w2 - epsi
lon_ary2)
                cost1 = self._get_cost(y_enc, a3, w1, w2 - epsil
on_ary2)
                a1, z2, a2, z3, a3 = self._feedforward(X, w1,
                                                       w2 + epsi
lon_ary2)
                cost2 = self._get_cost(y_enc, a3, w1, w2 + epsil
on_ary2)
                num_grad2[i, j] = (cost2 - cost1) / (2.0 * epsil
on)
                epsilon_ary2[i, j] = 0

        num_grad = np.hstack((num_grad1.flatten(), num_grad2.fla
tten()))
        grad = np.hstack((grad1.flatten(), grad2.flatten()))
        norm1 = np.linalg.norm(num_grad - grad)
        norm2 = np.linalg.norm(num_grad)
        norm3 = np.linalg.norm(grad)
        relative_error = norm1 / (norm2 + norm3)
        return relative_error

    def predict(self, X):
        """Predict class labels

        Parameters
        -----------
        X : array, shape = [n_samples, n_features]
            Input layer with original features.

        Returns:
        ----------
        y_pred : array, shape = [n_samples]
            Predicted class labels.

        """
```

```
        if len(X.shape) != 2:
            raise AttributeError('X must be a [n_samples, n_feat
ures] array.\n'
                                 'Use X[:,None] for 1-feature cl
assification,'
                                 '\nor X[[i]] for 1-sample class
ification')

        a1, z2, a2, z3, a3 = self._feedforward(X, self.w1, self.
w2)
        y_pred = np.argmax(z3, axis=0)
        return y_pred

    def fit(self, X, y, print_progress=False):
        """ Learn weights from training data.

        Parameters
        -----------
        X : array, shape = [n_samples, n_features]
            Input layer with original features.
        y : array, shape = [n_samples]
            Target class labels.
        print_progress : bool (default: False)
            Prints progress as the number of epochs
            to stderr.

        Returns:
        ----------
        self

        """
        self.cost_ = []
        X_data, y_data = X.copy(), y.copy()
        y_enc = self._encode_labels(y, self.n_output)

        delta_w1_prev = np.zeros(self.w1.shape)
        delta_w2_prev = np.zeros(self.w2.shape)

        for i in range(self.epochs):
```

```
            # adaptive learning rate
            self.eta /= (1 + self.decrease_const*i)

            if print_progress:
                sys.stderr.write('\rEpoch: %d/%d' % (i+1, self.e
pochs))

                sys.stderr.flush()

            if self.shuffle:
                idx = np.random.permutation(y_data.shape[0])
                X_data, y_enc = X_data[idx], y_enc[idx]

            mini = np.array_split(range(y_data.shape[0]), self.m
inibatches)
            for idx in mini:

                # feedforward
                a1, z2, a2, z3, a3 = self._feedforward(X[idx],
                                                       self.w1,
                                                       self.w2)
                cost = self._get_cost(y_enc=y_enc[:, idx],
                                      output=a3,
                                      w1=self.w1,
                                      w2=self.w2)
                self.cost_.append(cost)

                # compute gradient via backpropagation
                grad1, grad2 = self._get_gradient(a1=a1, a2=a2,
                                                  a3=a3, z2=z2,
                                                  y_enc=y_enc[:,
 idx],
                                                  w1=self.w1,
                                                  w2=self.w2)

                # start gradient checking
                grad_diff = self._gradient_checking(X=X_data[idx
],
                                                    y_enc=y_enc[
:, idx],
                                                    w1=self.w1,
```

```
                                                    w2=self.w2,
                                                    epsilon=1e-5
,
                                                    grad1=grad1,
                                                    grad2=grad2)



                if grad_diff <= 1e-7:
                    print('Ok: %s' % grad_diff)
                elif grad_diff <= 1e-4:
                    print('Warning: %s' % grad_diff)
                else:
                    print('PROBLEM: %s' % grad_diff)

                # update weights; [alpha * delta_w_prev] for mom
entum learning
                delta_w1, delta_w2 = self.eta * grad1, self.eta
* grad2
                self.w1 -= (delta_w1 + (self.alpha * delta_w1_pr
ev))
                self.w2 -= (delta_w2 + (self.alpha * delta_w2_pr
ev))
                delta_w1_prev, delta_w2_prev = delta_w1, delta_w
2

        return self
```

```
nn_check = MLPGradientCheck(n_output=10,
                            n_features=X_train.shape[1],
                            n_hidden=10,
                            l2=0.0,
                            l1=0.0,
                            epochs=10,
                            eta=0.001,
                            alpha=0.0,
                            decrease_const=0.0,
                            minibatches=1,
                            shuffle=False,
                            random_state=1)
```

```
nn_check.fit(X_train[:5], y_train[:5], print_progress=False)
```

```
Ok: 2.59699590792e-10
Ok: 2.9553528175e-10
Ok: 2.38060754028e-10
Ok: 3.07760791451e-10
Ok: 3.38742154283e-10
Ok: 3.57890531092e-10
Ok: 2.17697902147e-10
Ok: 2.36171066791e-10
Ok: 3.42158139292e-10
Ok: 2.10657747496e-10



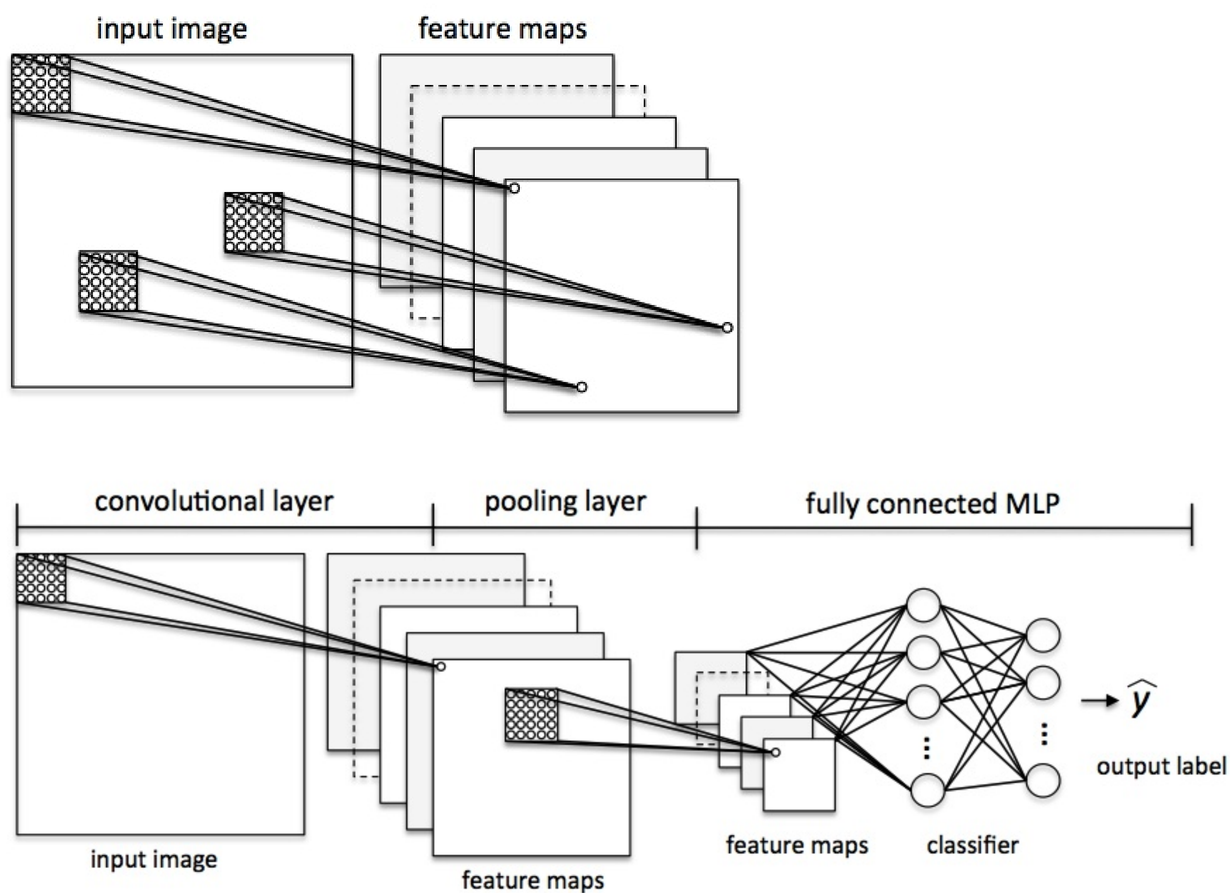

<__main__.MLPGradientCheck at 0x1288634d0>
```

Other neural network architectures

[back to top]

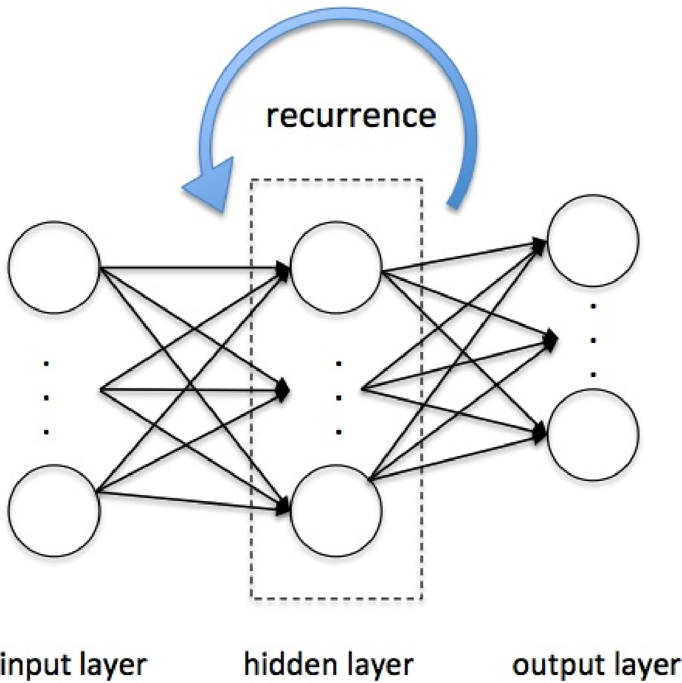
Convolutional Neural Networks

[back to top]



Recurrent Neural Networks

[back to top]


recurrence
input layer
hidden layer
output layer

Sections

Introduction to Deep Learning with TensorFlow

[back to top]

What is Deep Learning?

[back to top]

> *Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.*

I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning." Book in preparation for MIT Press, 2016.
http://www.deeplearningbook.org/

Representation Learning

> *Use machine learning to discover not only the mapping from representation to output but also the representation itself.*

I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning." Book in preparation for MIT Press, 2016.
http://www.deeplearningbook.org/

What is TensorFlow

[back to top]



> *TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.*

- A TensorFlow computation is described by a directed **graph**, which is composed of a set on **nodes**
- Each node represents the instantiation of an **operation**
- An **operation** represents an abstract computation (e.g., "matrix multiply", or "add")
- Clients programs interact with the TensorFlow system by creating a **Session**
- Computations represented as a dataflow graph where **tensors** flow along the graph edges

What is a Data Flow Graph

[back to top]

> *Data flow graphs describe mathematical computation with a directed graph of nodes & edges. Nodes typically implement mathematical operations, but can also represent endpoints to feed in data, push out results, or read/write persistent variables. Edges describe the input/output relationships between nodes. These data edges carry dynamically-sized multidimensional data arrays, or tensors. The flow of tensors through the graph is where TensorFlow gets its name. Nodes are assigned to computational devices and execute asynchronously and in parallel once all the tensors on their incoming edges becomes available.*

TensorFlow examples

THE EXTENT OF WHAT I KNOW ABOUT

TENSORFLOW IN ONE SLIDE

```
import numpy as np
import tensorflow as tf

X = tf.placeholder("float", [None, input_dim])
Y = tf.placeholder("float", [None, output_dim])

beta = tf.Variable(tf.random_normal(beta_shape, stddev=0.01))

def model(X, beta):
  # some function of X and beta

p_yx = model(X, beta)

cost = some_cost_function(p_yx, Y)
train_op = tf.train.SomeOptimizer.minimize(cost)

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())
  for _ in range(num_epochs):
    sess.run(train_op, feed_dict={X: trX, Y: trY})
```

STANDARD IMPORTS

PLACEHOLDERS FOR OUR DATA

PARAMETERS TO LEARN

SOME PARAMETRIC MODEL

APPLIED TO THE SYMBOLIC VARIABLES

TRAIN BY MINIMIZING SOME COST FUNCTION

CREATE SESSION AND INITIALIZE VARIABLES

TRAIN USING DATA

[back to top]

Introduction

[back to top]

Hello world

[back to top]

- Session: TensorFlow 是在 session 中运行 computation graph
- Fetches: 在 session 中执行 run()， 可以 fetch 得到 operation 的结果

```python
# Simple hello world using TensorFlow
import tensorflow as tf


# Create a Constant op
# The op is added as a node to the default graph.
#
# The value returned by the constructor represents the output
# of the Constant op.

hello = tf.constant('Hello, TensorFlow!')

# Start tf session
sess = tf.Session()

# Run graph
print sess.run(hello)
```

```
Hello, TensorFlow!
```

Basic Operations

[back to top]

```
import tensorflow as tf

# Basic constant operations
# The value returned by the constructor represents the output
# of the Constant op.
a = tf.constant(2)
b = tf.constant(3)

# Launch the default graph.
with tf.Session() as sess:
    print "a=2, b=3"
    print "Addition with constants: %i" % sess.run(a+b)
    print "Multiplication with constants: %i" % sess.run(a*b)
```

```
a=2, b=3
Addition with constants: 5
Multiplication with constants: 6
```

- Feed: 在 sess.run() 中传入 feed_dict 参数可以向对应的节点喂入数据
- placeholder 作为一个占位符，是数据输入的端点，必须要在 run() 中喂入数据

```
# Basic Operations with variable as graph input
# The value returned by the constructor represents the output
# of the Variable op. (define as input when running session)
# tf Graph input
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)
# placeholder 是信息输入的端点
# 在 sess 中通过 feed_dict 参数来喂入数据

# Define some operations
add = tf.add(a, b)
mul = tf.mul(a, b)
# 这里定义的操作是象征性的，sess.run 之后才会真正在内存中跑起来

# Launch the default graph.
with tf.Session() as sess:
    # Run every operation with variable input
    print "Addition with variables: %i" % sess.run(add, feed_dic
t={a: 2, b: 3})
    print "Multiplication with variables: %i" % sess.run(mul, fe
ed_dict={a: 2, b: 3})
```

```
Addition with variables: 5
Multiplication with variables: 6
```

```
# ----------------
# More in details:
# Matrix Multiplication from TensorFlow official tutorial

# Create a Constant op that produces a 1x2 matrix.  The op is
# added as a node to the default graph.
#
# The value returned by the constructor represents the output
# of the Constant op.
matrix1 = tf.constant([[3., 3.]])

# Create another Constant that produces a 2x1 matrix.
```

```
matrix2 = tf.constant([[2.],[2.]])

# Create a Matmul op that takes 'matrix1' and 'matrix2' as input
s.
# The returned value, 'product', represents the result of the ma
trix
# multiplication.
product = tf.matmul(matrix1, matrix2)

# To run the matmul op we call the session 'run()' method, passi
ng 'product'
# which represents the output of the matmul op.  This indicates
to the call
# that we want to get the output of the matmul op back.
#
# All inputs needed by the op are run automatically by the sessi
on.  They
# typically are run in parallel.
#
# The call 'run(product)' thus causes the execution of threes op
s in the
# graph: the two constants and matmul.
#
# The output of the op is returned in 'result' as a numpy `ndarr
ay` object.
with tf.Session() as sess:
    result = sess.run(product)
    print result
```

```
[[ 12.]]
```

Basic Models

Linear Regression

[back to top]

Variable:

- variable 是在计算图中可训练的量
- 在 session 中必须先要初始化
- name 参数可定义 variable 在 graph 中的名称

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Parameters
learning_rate = 0.01
training_epochs = 1000
display_step = 100
```

```
# Training Data
train_X = np.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                         7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = np.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                         2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]
```

```
# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")
```

```
# Construct a linear model
pred = tf.add(tf.mul(X, W), b)
```

```
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
# Gradient descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).min
imize(cost)
```

```
# Initializing the variables
init = tf.initialize_all_variables() # 定义初始化操作
```

```
# Launch the graph
with tf.Session() as sess:
    sess.run(init)  # variable 在 session 中必须先初始化

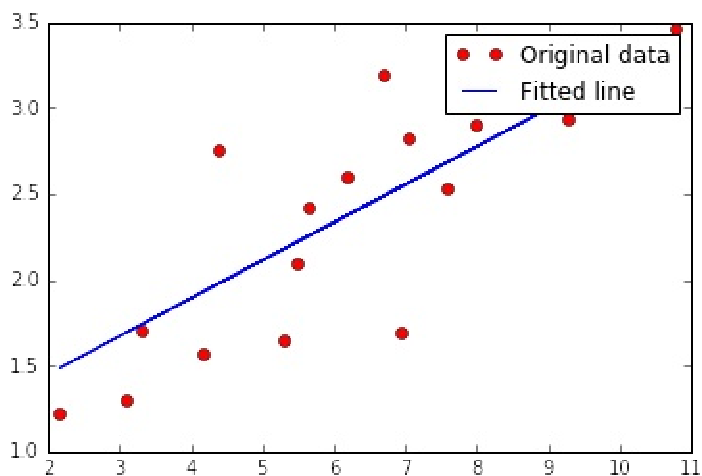
    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})  # 每 ru
n 一次 optimizer， 进行一次梯度下降

        #Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, Y:train_Y}
)
            print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}"
.format(c), \
                "W=", sess.run(W), "b=", sess.run(b)

    print "Optimization Finished!"
    training_cost = sess.run(cost, feed_dict={X: train_X, Y: tra
in_Y})
    print "Training cost=", training_cost, "W=", sess.run(W), "b
=", sess.run(b), '\n'

    #Graphic display
    plt.plot(train_X, train_Y, 'ro', label='Original data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label=
'Fitted line')
    plt.legend()
    plt.show()
```

```
Epoch: 0050 cost= 0.104400784 W= 0.157368 b= 1.46493
Epoch: 0100 cost= 0.101245113 W= 0.162853 b= 1.42547
Epoch: 0150 cost= 0.098453194 W= 0.168012 b= 1.38836
Epoch: 0200 cost= 0.095982887 W= 0.172864 b= 1.35345
Epoch: 0250 cost= 0.093797326 W= 0.177427 b= 1.32062
Epoch: 0300 cost= 0.091863610 W= 0.18172 b= 1.28975
Epoch: 0350 cost= 0.090152740 W= 0.185757 b= 1.26071
Epoch: 0400 cost= 0.088638850 W= 0.189554 b= 1.23339
Epoch: 0450 cost= 0.087299488 W= 0.193125 b= 1.2077
Epoch: 0500 cost= 0.086114518 W= 0.196483 b= 1.18354
Epoch: 0550 cost= 0.085066028 W= 0.199641 b= 1.16082
Epoch: 0600 cost= 0.084138311 W= 0.202612 b= 1.13945
Epoch: 0650 cost= 0.083317406 W= 0.205406 b= 1.11935
Epoch: 0700 cost= 0.082590967 W= 0.208034 b= 1.10044
Epoch: 0750 cost= 0.081948124 W= 0.210505 b= 1.08266
Epoch: 0800 cost= 0.081379279 W= 0.21283 b= 1.06594
Epoch: 0850 cost= 0.080875866 W= 0.215017 b= 1.05021
Epoch: 0900 cost= 0.080430366 W= 0.217073 b= 1.03542
Epoch: 0950 cost= 0.080036096 W= 0.219007 b= 1.0215
Epoch: 1000 cost= 0.079687178 W= 0.220826 b= 1.00842
Optimization Finished!
Training cost= 0.0796872 W= 0.220826 b= 1.00842
```



Logistic Regression

[back to top]

```
import tensorflow as tf

# Import MINST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

```
# Parameters
learning_rate = 0.01
training_epochs = 25
batch_size = 100
display_step = 5

# tf Graph Input
x = tf.placeholder(tf.float32, [None, 784]) # mnist data image of shape 28*28=784
y = tf.placeholder(tf.float32, [None, 10]) # 0-9 digits recognition => 10 classes

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# Construct model
pred = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))  # tf.reduce_mean 类似 np.mean()
# Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initializing the variables
init = tf.initialize_all_variables()
```

```
# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_si
ze)
            # Fit training using batch data
            _, c = sess.run([optimizer, cost], feed_dict={x: bat
ch_xs,
                                                          y: bat
ch_ys})
            # Compute average loss
            avg_cost += c / total_batch
        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}"
.format(avg_cost)

    print "Optimization Finished!"

    # Test model
    correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(
y, 1))
    # Calculate accuracy for 3000 examples
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.flo
at32))
    print "Accuracy:", accuracy.eval({x: mnist.test.images[:3000
], y: mnist.test.labels[:3000]})
```

```
Epoch: 0005 cost= 0.465507779
Epoch: 0010 cost= 0.392393045
Epoch: 0015 cost= 0.362739271
Epoch: 0020 cost= 0.345433382
Epoch: 0025 cost= 0.333723887
Optimization Finished!
Accuracy: 0.888333
```

Neural Networks

[back to top]

Multilayer Perceptron

[back to top]

```
# Import MINST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

import tensorflow as tf
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

```
# Parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
display_step = 5

# Network Parameters
n_hidden_1 = 256 # 1st layer number of features
n_hidden_2 = 256 # 2nd layer number of features
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10 # MNIST total classes (0-9 digits)

# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])
```

```
# Create model
def multilayer_perceptron(x, weights, biases):
    # Hidden layer with RELU activation
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)
    # Hidden layer with RELU activation
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['
b2'])
    layer_2 = tf.nn.relu(layer_2)
    # Output layer with linear activation
    out_layer = tf.matmul(layer_2, weights['out']) + biases['out'
]
    return out_layer
```

```
# Store layers weight & bias
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])
),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes])
)
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}

# Construct model
pred = multilayer_perceptron(x, weights, biases)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).
minimize(cost)

# Initializing the variables
init = tf.initialize_all_variables()
```

```
# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_x, batch_y = mnist.train.next_batch(batch_size
)
            # Run optimization op (backprop) and cost op (to get
 loss value)
            _, c = sess.run([optimizer, cost], feed_dict={x: bat
ch_x,
                                                          y: bat
ch_y})
            # Compute average loss
            avg_cost += c / total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
            print "Epoch:", '%04d' % (epoch+1), "cost=", \
                "{:.9f}".format(avg_cost)
    print "Optimization Finished!"

    # Test model
    correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(
y, 1))
    # Calculate accuracy
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"
))
    print "Accuracy:", accuracy.eval({x: mnist.test.images, y: m
nist.test.labels})
```

```
Epoch: 0001 cost= 166.660608705
Epoch: 0006 cost= 9.265776215
Epoch: 0011 cost= 2.211729868
Optimization Finished!
Accuracy: 0.9434
```

Convolutional Neural Network

[back to top]

```
import tensorflow as tf

# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

```
# Parameters
learning_rate = 0.001
training_iters = 200000
batch_size = 128
display_step = 200

# Network Parameters
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10 # MNIST total classes (0-9 digits)
dropout = 0.75 # Dropout, probability to keep units

# tf Graph input
x = tf.placeholder(tf.float32, [None, n_input])
y = tf.placeholder(tf.float32, [None, n_classes])
keep_prob = tf.placeholder(tf.float32) #dropout (keep probabilit
y)
```

```
# Create some wrappers for simplicity
def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], pad
ding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)


def maxpool2d(x, k=2):
    # MaxPool2D wrapper
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k,
k, 1],
                          padding='SAME')


# Create model
def conv_net(x, weights, biases, dropout):
    # Reshape input picture
    x = tf.reshape(x, shape=[-1, 28, 28, 1])
```

```
    # Convolution Layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling)
    conv1 = maxpool2d(conv1, k=2)

    # Convolution Layer
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling)
    conv2 = maxpool2d(conv2, k=2)

    # Fully connected layer
    # Reshape conv2 output to fit fully connected layer input
    fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_l
ist()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)
    # Apply Dropout
    fc1 = tf.nn.dropout(fc1, dropout)

    # Output, class prediction
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
    return out
```

```
# Store layers weight & bias
weights = {
    # 5x5 conv, 1 input, 32 outputs
    'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])),
    # 5x5 conv, 32 inputs, 64 outputs
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    # fully connected, 7*7*64 inputs, 1024 outputs
    'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])),
    # 1024 inputs, 10 outputs (class prediction)
    'out': tf.Variable(tf.random_normal([1024, n_classes]))
}

biases = {
    'bc1': tf.Variable(tf.random_normal([32])),
    'bc2': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}

# Construct model
pred = conv_net(x, weights, biases, keep_prob)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).
minimize(cost)

# Evaluate model
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initializing the variables
init = tf.initialize_all_variables()
```

```
# Launch the graph
with tf.Session() as sess:
    sess.run(init)
    step = 1
    # Keep training until reach max iterations
    while step * batch_size < training_iters:
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop)
        sess.run(optimizer, feed_dict={x: batch_x, y: batch_y,
                                       keep_prob: dropout})
        if step % display_step == 0:
            # Calculate batch loss and accuracy
            loss, acc = sess.run([cost, accuracy], feed_dict={x:
 batch_x,
                                                              y:
 batch_y,
                                                              ke
ep_prob: 1.})
            print "Iter " + str(step*batch_size) + ", Minibatch
Loss= " + \
                  "{:.6f}".format(loss) + ", Training Accuracy= "
 + \
                  "{:.5f}".format(acc)
        step += 1
    print "Optimization Finished!"

    # Calculate accuracy for 256 mnist test images
    print "Testing Accuracy:", \
        sess.run(accuracy, feed_dict={x: mnist.test.images[:256]
,
                                      y: mnist.test.labels[:256]
,
                                      keep_prob: 1.})
```

```
Iter 25600, Minibatch Loss= 1453.969238, Training Accuracy= 0.87
500
Iter 51200, Minibatch Loss= 0.000000, Training Accuracy= 1.00000
Iter 76800, Minibatch Loss= 836.579651, Training Accuracy= 0.914
06
Iter 102400, Minibatch Loss= 265.563293, Training Accuracy= 0.96
875
Iter 128000, Minibatch Loss= 120.997910, Training Accuracy= 0.99
219
Iter 153600, Minibatch Loss= 29.434311, Training Accuracy= 0.976
56
Iter 179200, Minibatch Loss= 248.191101, Training Accuracy= 0.98
438
Optimization Finished!
Testing Accuracy: 0.984375
```

Recurrent Neural Network LSTM

[back to top]

```
import tensorflow as tf
from tensorflow.python.ops import rnn, rnn_cell
import numpy as np

# Import MINST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

```
# Parameters
learning_rate = 0.001
training_iters = 100000
batch_size = 128
display_step = 100

# Network Parameters
n_input = 28 # MNIST data input (img shape: 28*28)
n_steps = 28 # timesteps
n_hidden = 128 # hidden layer num of features
n_classes = 10 # MNIST total classes (0-9 digits)

# tf Graph input
x = tf.placeholder("float", [None, n_steps, n_input])
y = tf.placeholder("float", [None, n_classes])

# Define weights
weights = {
    'out': tf.Variable(tf.random_normal([n_hidden, n_classes]))
}
biases = {
    'out': tf.Variable(tf.random_normal([n_classes]))
}
```

```
def RNN(x, weights, biases):

    # Prepare data shape to match `rnn` function requirements
    # Current data input shape: (batch_size, n_steps, n_input)
    # Required shape: 'n_steps' tensors list of shape (batch_siz
e, n_input)

    # Permuting batch_size and n_steps
    x = tf.transpose(x, [1, 0, 2])
    # Reshaping to (n_steps*batch_size, n_input)
    x = tf.reshape(x, [-1, n_input])
    # Split to get a list of 'n_steps' tensors of shape (batch_s
ize, n_input)
    x = tf.split(0, n_steps, x)
```

```
    # Define a lstm cell with tensorflow
    lstm_cell = rnn_cell.BasicLSTMCell(n_hidden, forget_bias=1.0
, state_is_tuple=True)

    # Get lstm cell output
    outputs, states = rnn.rnn(lstm_cell, x, dtype=tf.float32)

    # Linear activation, using rnn inner loop last output
    return tf.matmul(outputs[-1], weights['out']) + biases['out'
]

pred = RNN(x, weights, biases)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pr
ed, y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).
minimize(cost)

# Evaluate model
correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initializing the variables
init = tf.initialize_all_variables()
```

```
# Launch the graph
with tf.Session() as sess:
    sess.run(init)
    step = 1
    # Keep training until reach max iterations
    while step * batch_size < training_iters:
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Reshape data to get 28 seq of 28 elements
        batch_x = batch_x.reshape((batch_size, n_steps, n_input)
)
        # Run optimization op (backprop)
        sess.run(optimizer, feed_dict={x: batch_x, y: batch_y})
        if step % display_step == 0:
            # Calculate batch accuracy
            acc = sess.run(accuracy, feed_dict={x: batch_x, y: b
atch_y})
            # Calculate batch loss
            loss = sess.run(cost, feed_dict={x: batch_x, y: batc
h_y})
            print "Iter " + str(step*batch_size) + ", Minibatch
Loss= " + \
                  "{:.6f}".format(loss) + ", Training Accuracy= "
 + \
                  "{:.5f}".format(acc)
        step += 1
    print "Optimization Finished!"

    # Calculate accuracy for 128 mnist test images
    test_len = 128
    test_data = mnist.test.images[:test_len].reshape((-1, n_step
s, n_input))
    test_label = mnist.test.labels[:test_len]
    print "Testing Accuracy:", \
        sess.run(accuracy, feed_dict={x: test_data, y: test_labe
l})
```

```
Iter 12800, Minibatch Loss= 0.716396, Training Accuracy= 0.75000
Iter 25600, Minibatch Loss= 0.367348, Training Accuracy= 0.87500
Iter 38400, Minibatch Loss= 0.164333, Training Accuracy= 0.92969
Iter 51200, Minibatch Loss= 0.143476, Training Accuracy= 0.92969
Iter 64000, Minibatch Loss= 0.193304, Training Accuracy= 0.96094
Iter 76800, Minibatch Loss= 0.202645, Training Accuracy= 0.90625
Iter 89600, Minibatch Loss= 0.056868, Training Accuracy= 0.98438
Optimization Finished!
Testing Accuracy: 0.992188
```

使用卷积神经网络做文本分类

```
from os import path
import os
import re
import codecs
import pandas as pd
import numpy as np
```

```
from cPickle import dump,load
#dump(df, open('data/tmdf.pickle', 'wb'))
df = load(open('data/tmdf.pickle','rb'))
```

```
df.head()
```

	label	txt	seg_word
0	0	本报记者陈雪频实习记者唐翔发自上海\r\n 一家刚刚成立两年的网络支付公司，它的目标是...	本报记者 陈雪频 实习 记者 唐翔 发自 上海 \r\n 一家 刚刚 成立 ...
1	0	证券通：百联股份未来5年有能力保持高速增长\r\n\r\n 深度报告 权威内参...	证券 通 ： 百联 股份 未来 5 年 有 能力 保持高速 增长 \r\n ...
2	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....	5 月 09 日 消息 快评 \r\n \r\n 深度 报告...
3	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....	5 月 09 日 消息 快评 \r\n \r\n 深度 报告...
4	0	5月09日消息快评\r\n\r\n 深度报告 权威内参 来自“证券通”www....	5 月 09 日 消息 快评 \r\n \r\n 深度 报告...

```
#  文本整理完毕，后面建模需要将词汇转成数字编号，可以人工转，也可以让keras转

textraw = df.seg_word.values.tolist()
textraw = [line.encode('utf-8') for line in textraw] # 需要存为str才能被keras使用
```

```
maxfeatures = 50000 # 只选择最重要的词
from keras.preprocessing.text import Tokenizer
token = Tokenizer(nb_words=maxfeatures)
token.fit_on_texts(textraw) #如果文本较大可以使用文本流
text_seq = token.texts_to_sequences(textraw)
```

```
np.median([len(x) for x in text_seq]) #  每条新闻平均400个词汇
```

```
498.0
```

```
y = df.label.values # 定义好标签
nb_classes = len(np.unique(y))
print(nb_classes)
```

```
9
```

```
from __future__ import absolute_import
from keras.optimizers import RMSprop
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatte
n
from keras.layers.embeddings import Embedding
from keras.layers.convolutional import Convolution1D, MaxPooling
1D
from keras.layers.recurrent  import SimpleRNN, GRU, LSTM
from keras.callbacks import EarlyStopping
```

```
maxlen = 600 # 定义文本最大长度
batch_size = 32 # 批次
word_dim = 100 # 词向量维度
nb_filter = 200  # 卷积核个数
filter_length = 10 # 卷积窗口大小
hidden_dims = 50  # 隐藏层神经元个数
nb_epoch = 10      # 训练迭代次数
pool_length = 50   # 池化窗口大小
```

```
from sklearn.cross_validation import train_test_split
train_X, test_X, train_y, test_y = train_test_split(text_seq, y
, train_size=0.8, random_state=1)
```

```
# 转为等长矩阵，长度为maxlen
print("Pad sequences (samples x time)")
X_train = sequence.pad_sequences(train_X, maxlen=maxlen,padding=
'post', truncating='post')
X_test = sequence.pad_sequences(test_X, maxlen=maxlen,padding='p
ost', truncating='post')
print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)
```

```
Pad sequences (samples x time)
('X_train shape:', (14328, 600))
('X_test shape:', (3582, 600))
```

```
# 将y的格式展开成one-hot
from keras.utils import np_utils
Y_train = np_utils.to_categorical(train_y, nb_classes)
Y_test = np_utils.to_categorical(test_y, nb_classes)
```

```
# for version bug
import tensorflow as tf
tf.python.control_flow_ops = tf
```

```
# CNN 模型
print('Build model...')
model = Sequential()

# 词向量嵌入层，输入：词典大小，词向量大小，文本长度
model.add(Embedding(maxfeatures, word_dim,input_length=maxlen,dr
opout=0.25))
model.add(Convolution1D(nb_filter=nb_filter,
                        filter_length=filter_length,
                        border_mode="valid",
                        activation="relu"))
# 池化层
model.add(MaxPooling1D(pool_length=pool_length))
model.add(Flatten())
# 全连接层
model.add(Dense(hidden_dims))
model.add(Dropout(0.25))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmspro
p',metrics=["accuracy"])
```

```
Build model...
```

```
earlystop = EarlyStopping(monitor='val_loss', patience=1, verbos
e=1)
result = model.fit(X_train, Y_train, batch_size=batch_size, nb_e
poch=nb_epoch,
            validation_split=0.1, callbacks=[earlystop])
```

```
/Users/xiaokai/anaconda/envs/tensorflow/lib/python2.7/site-packa
ges/keras/models.py:603: UserWarning: The "show_accuracy" argume
nt is deprecated, instead you should pass the "accuracy" metric
to the model at compile time:
`model.compile(optimizer, loss, metrics=["accuracy"])`
  warnings.warn('The "show_accuracy" argument is deprecated, '
/Users/xiaokai/anaconda/envs/tensorflow/lib/python2.7/site-packa
ges/tensorflow/python/ops/gradients.py:90: UserWarning: Converti
ng sparse IndexedSlices to a dense Tensor of unknown shape. This
 may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown
shape. "



Train on 12895 samples, validate on 1433 samples
Epoch 1/10
12895/12895 [==============================] - 704s - loss: 1.43
06 - val_loss: 0.5532
Epoch 2/10
12895/12895 [==============================] - 7724s - loss: 0.4
912 - val_loss: 0.4273
Epoch 3/10
12895/12895 [==============================] - 765s - loss: 0.35
11 - val_loss: 0.4003
Epoch 4/10
12895/12895 [==============================] - 807s - loss: 0.25
71 - val_loss: 0.4114
Epoch 5/10
12864/12895 [============================>.] - ETA: 5s - loss: 0
.1971 Epoch 00004: early stopping
12895/12895 [==============================] - 2285s - loss: 0.1
968 - val_loss: 0.4415
```

```
score = earlystop.model.evaluate(X_test, Y_test, batch_size=batc
h_size)
print('Test score:', score)
classes = earlystop.model.predict_classes(X_test, batch_size=bat
ch_size)
acc = np_utils.accuracy(classes, test_y) # 要用没有转换前的y
print('Test accuracy:', acc)
```

```
3582/3582 [==============================] - 73s
('Test score:', 0.4292584941548252)
3582/3582 [==============================] - 73s
('Test accuracy:', 0.89056393076493578)
```